J.P.A. van Prooijen

BASIC VOOR BEGINNERS

Kluwer Technische Boeken

BASIC voor beginners

J. P. A. van Prooijen

BASIC VOOR BEGINNERS

Een inleiding tot het programmeren
in de computertaal BASIC



Kluwer Technische Boeken B.V. Deventer-Antwerpen

Redactie: P. P. Mastboom

ISBN 90 201 1257 0
D/1981/0108/177

1e druk 1980
2e oplage 1981
3e oplage 1981
4e oplage 1983

Woord vooraf

De programmeertaal BASIC is oorspronkelijk ontwikkeld aan het Dartmouth College in de Verenigde Staten. Vooral de laatste jaren zijn, ook ten gevolge van de opkomst van de vele typen microcomputers, door diverse fabrikanten wijzigingen aangebracht in de „standaard-BASIC", ten einde de unieke eigenschappen van de betreffende computers ook in BASIC te kunnen benutten.
Een en ander heeft tot gevolg gehad dat er inmiddels vele BASIC-versies in omloop zijn, die ten opzichte van elkaar meer of minder verschillen. Het zou ondoenlijk zijn om op alle BASIC-versies in te gaan, zodat is besloten om in dit boek „standaard-BASIC" te bespreken. Heeft U dat eenmaal onder de knie, dan is het een kleine moeite om, met behulp van de documentatie bij de computer, de specifieke eigenschappen van Uw BASIC-versie te leren kennen.

Inhoud

1. Inleiding

Wanneer we een computer een bepaalde opdracht uit willen laten voeren, moeten we hem een programma aanbieden, dat deze opdracht stap voor stap beschrijft. Elke stap noemen we een instructie; alle instructies samen vormen dus een programma.
De instructies moeten in een speciale zgn. programmeertaal worden geschreven. Voor deze programmeertaal hebben we in het algemeen een reeks van mogelijkheden, die we echter in 2 hoofdgroepen kunnen verdelen, nl. machinegerichte programmeertalen en niet-machinegerichte of hogere programmeertalen.

Schrijven we het programma in een *machinegerichte taal*, dan kan dit programma slechts op 1 type computer worden uitgevoerd. De interne opbouw of architectuur is nl. van computer tot computer verschillend. Elke computer heeft dan ook zijn eigen instructierepertoire, zodat een programma voor een bepaalde opdracht er voor elke computer anders uitziet.
Overgang naar een ander type computer betekent herschrijven van het programma. We kunnen eigenlijk zeggen, dat een machinegerichte taal door de fabrikant is vastgesteld.
Een *niet-machinegerichte of hogere programmeertaal* staat a.h.w. ,,boven de machine". Bij een hogere programmeertaal zijn we niet meer gebonden aan het type computer. Bovendien weerspiegelt een dergelijk programma beter de oplossing, omdat 1 instructie in een hogere programmeertaal (dit noemen we een *statement*) uiteindelijk resulteert in een aantal opdrachten voor de computer. Bij een machinegerichte taal is dit niet het geval; hier resulteert 1 instructie in 1 opdracht voor de computer. Het is dan ook mogelijk om de instructies van een machinegerichte taal direct in énen en nullen te schrijven en deze codes aan de computer aan te bieden. Bij een hogere programmeertaal is dit niet mogelijk. Hierbij moet elke statement eerst worden vertaald in een aantal codes voor de betreffende computer.
Dit vertaalproces kan door de computer zelf worden uitgevoerd m.b.v. een vertaalprogramma, *compiler* genaamd. Zo'n compiler is dus wel afhankelijk van het type computer en is dan ook meestal bij de fabrikant van de betreffende computer verkrijgbaar.
U zult begrijpen, dat zo'n compiler een behoorlijk complex en lang programma is. Bovendien moet elk programma dat een computer moet uitvoeren, en dus

ook een compiler, in het geheugen van die computer worden opgeslagen. Zo'n compiler neemt dus veel geheugenruimte in beslag en dat is nu juist het probleem bij microcomputers.
In verhouding tot de microprocessor (het hart van de microcomputer) is geheugenruimte zeer duur, zodat men eigenlijk nog maar weinig microcomputersystemen ziet die bijv. in de hogere programmeertalen COBOL of ALGOL kunnen worden geprogrammeerd.

Er is echter een vrij eenvoudige hogere programmeertaal, die niet zoveel geheugenruimte voor het vertaalprogramma nodig heeft, en dat is de programmeertaal *BASIC (Beginners All purpose Symbolic Instruction Code).* Voor deze programmeertaal wordt meestal ook van een ietwat bijzondere compiler gebruik gemaakt, nl. een die steeds één instructie vertaalt in de vereiste codes en deze dan direct aan de processor aanbiedt om ze te laten uitvoeren, dan de volgende instructie vertaalt, enz. Een dergelijk vertaalprogramma heet een *interpreter* en we zeggen wel dat de taal BASIC interactief is, d.w.z. er bestaat een wisselwerking tussen het programma dat wij de computer aanbieden (via het toetsenbord) en de interpreter die de instructies van ons programma vertaalt en zorgt dat ze door de computer worden uitgevoerd. (Bij de andere hogere programmeertalen, zoals COBOL, FORTRAN en ALGOL is het zo, dat de compiler eerst alle door ons ingevoerde instructies vertaalt in codes en deze codes bijv. op een ponsband uitvoert. Daarna is de compiler niet meer nodig en kan de geheugenruimte die de compiler in beslag neemt worden gebruikt voor de opslag van bijv. het vertaalde programma).

2. BASIC-variaties

De programmeertaal BASIC, zoals die meestal wordt gebruikt en is gestandaardiseerd, is ontwikkeld op het Dartmouth College in de Verenigde Staten. De interpreter voor deze *standaard-BASIC* beslaat bij een (8 bit) microcomputer ca. 8K × 8 bit geheugenruimte (1K is in de computerwereld 1024. 8K × 8 bit betekent dat er 8 × 1024 = 8192 geheugenplaatsen zijn die elk 8 bits kunnen bevatten).
Er zijn ook verschillende *tiny-BASIC*-interpreters in omloop, die speciaal voor microcomputers zijn ontwikkeld. De tiny-BASIC-statements vormen een subset van de standaard BASIC en de interpreter beslaat ca. 2 tot 4K × 8 bit geheugenruimte.
Diverse fabrikanten van microcomputersystemen hebben meerdere statements aan de standaard BASIC toegevoegd, om de mogelijkheden van hun computersystemen ten volle te kunnen benutten. We spreken in dit geval van *extended (uitgebreide) BASIC* en de geheugenruimte die voor een dergelijke interpreter nodig is kan oplopen tot ca. 16K × 8 bit.
In dit boek zullen we ons in de eerste plaats bezighouden met standaard BASIC, dus de taal zoals die op het Dartmouth College is gestandaardiseerd.
Bovendien worden enkele instructies behandeld die voorkomen bij de extended-BASIC van de meeste „personal computers", zoals de PET, de TRS-80 en de APPLE.

Alle programma's die in dit boek voorkomen zijn getest op een PET-microcomputer met 32 Kbyte geheugen. De programmalistings zijn gemaakt met een Centronics 779 printer.

3. Algemene vorm van een BASIC-statement

Fig. 1 toont de algemene vorm van een BASIC-statement. Elke statement moet beginnen met een regelnummer. Daarna typen we een spatie in en dan de eigenlijke opdracht. In fig. 1 is dit de opdracht PRINT (to print = afdrukken). Deze opdracht geeft dus aan welke bewerking moet worden uitgevoerd. Na weer een spatie te hebben ingetypt, geven we aan waarmee de bewerking moet worden uitgevoerd. In fig. 1 hebben we de letter A ingetypt en met PRINT A bedoelen we, dat de waarde die de variabele A op dat moment heeft, moet worden afgedrukt. (Een variabele is een grootheid die van waarde kan veranderen.)

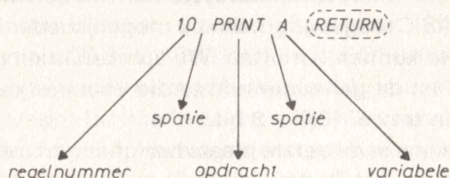


Fig. 1

Het einde van een statement geven we aan door op de toets RETURN (to return = terugkeren) te drukken. Wanneer we bijv. via een toetsenbord en beeldscherm met de computer communiceren, gaat na deze RETURN-opdracht de cursor op het beeldscherm terug naar het begin van de regel. De interpreter geeft als antwoord op de RETURN-opdracht het commando LINE FEED (= volgende regel). (Een cursor is een tekentje op het beeldscherm, dat aangeeft waar het volgende karakter terecht zal komen).

Werken we met bijv. een Teletype, dit is een combinatie van toetsenbord en printer (regeldrukker), dan gaat t.g.v. de RETURN-opdracht de „wagen" van de printer terug naar het begin van de regel.

De spaties tussen de verschillende delen van een statement zijn niet perse noodzakelijk. Zonder spaties begrijpt de interpreter meestal ook wel wat we bedoelen. Om echter het programma overzichtelijk te houden en om het achteraf nog eens te kunnen controleren, wordt aangeraden wel spaties te gebruiken.

Nog even iets over de regelnummers. Deze mogen in principe willekeurig worden gekozen, als we er maar rekening mee houden dat het programma altijd

in *oplopende nummervolgorde* wordt uitgevoerd (tenzij anders aangegeven door een spronginstructie).
In de praktijk nummeren we de regels met 10 toenemend. Dit is niet alleen zeer overzichtelijk, maar bovendien hebben we dan achteraf de mogelijkheid om nog statements toe te voegen, als we merken dat we iets vergeten zijn. Wanneer we de statements nummeren met 1, 2, 3, 4 enz. is dit natuurlijk niet mogelijk, omdat er geen regelnummers meer beschikbaar zijn.

4. List

Stel, we typen het volgende programma in (de betekenis van de statements volgt later):

```
10 LET B = 4
20 PRINT B
30 IF B < 10 THEN GOTO 20
40 END
```

en ontdekken nu, dat we tussen regel 20 en regel 30 nog een statement willen toevoegen, bijv. LET B = B + 1. We kunnen dit dan doen onder één van de nummers 21 t/m 29. Meestal kiezen we in zo'n geval het „midden", dus nummer 25 (voor het geval er nog meer missers zijn gemaakt). We typen dus in:

```
25 LET B = B + 1
```

Een afdruk van het totale programma dat zich nu in het geheugen bevindt, krijgen we als we het woord LIST intypen *en daarna op de RETURN-toets drukken.*

```
LIST
10 LET B = 4
20 PRINT B
25 LET B = B + 1
30 IF B < 10 THEN GOTO 20
40 END
```

U ziet, dat de interpreter de nieuwe statement keurig op de juiste plaats heeft ingevoegd.
We merken nog op, dat we elk programma moeten afsluiten met de END-statement, zoals in bovenstaande programma's in regel 40 is gedaan.

5. Run

Tijdens het intypen van het programma gedraagt de interpreter zich als een soort redacteur (editor). De letters, cijfers en tekens, die wij intypen worden in het geheugen opgeslagen, later ingetypte statements worden op de juiste plaats ingevoegd, syntaxfouten (dit zijn fouten in de opbouw van een statement, bijv. typefouten) worden gedetecteerd, enz. Er worden echter nog geen statements vertaald of uitgevoerd. Dit gebeurt pas als we het woord RUN intypen en daarna op de toets RETURN drukken. Dan worden de statements achtereenvolgens vertaald en uitgevoerd.

De uitvoering van een programma kan op 3 manieren worden beëindigd, nl. wanneer:

1. alle statements zijn uitgevoerd;
2. een bewerking moet worden uitgevoerd die niet mogelijk is, bijv. doordat de max. getalgrootte wordt overschreden;
3. tegelijkertijd op de toetsen CTRL en C wordt gedrukt (bij sommige interpreters geldt dit ook voor de toetsen BREAK of ESCAPE).

We zien, dat we te maken hebben met 2 soorten opdrachten of moden, nl. de *executie-mode* en de *definitie-mode*. Tot de executie-mode behoren de „normale" BASIC-statements, zoals PRINT, LET en GOTO; tot de definitiemode horen opdrachten zoals LIST en RUN.

6. NEW

Omdat het mogelijk is om achteraf nog statements toe te voegen met afwijkende regelnummers bestaat het gevaar, dat zich tussen de statements van een nieuw programma nog statements van een vorig programma bevinden.
Stel, dat zich in het geheugen van de computer het volgende programma bevindt:

```
10 LET B = 10
20 PRINT B
25 LET B = B – 1
30 IF B < 0 THEN GOTO 20
40 END
```

en dat we nu een geheel ander programma intypen, bijv.:

```
10 LET B = 3 + 7
20 PRINT B
30 LET A = 20 + B
40 PRINT A
50 END
```

We zien, dat de regels 10, 20, 30 en 40 van het oude programma zijn *overschreven* met nieuwe statements.
Regel 25 echter niet, zodat de statement LET B = B – 1 zich nog in het geheugen bevindt. Wanneer we nu een listing opvragen, dan krijgen we het volgende:

```
LIST
10 LET B = 3 + 7
20 PRINT B
25 LET B = B – 1
30 LET A = 20 + B
40 PRINT A
50 END
```

Dergelijke fouten kunnen we voorkomen, door er een gewoonte van te maken, dat we aan het begin van *ieder* programma eerst het woord NEW intypen en daarna op de RETURN-toets drukken. T.g.v. deze opdracht wordt nl. het gehele geheugen gewist (behalve natuurlijk dat deel waarin zich de interpreter bevindt...).

7. Rekenen

Wanneer we de computer rekenkundige bewerkingen willen laten uitvoeren hebben we de keuze uit:

1. machtsverheffen (↑)
2. vermenigvuldigen (*)
3. delen (/)
4. optellen (+)
5. aftrekken (−)

Wanneer in een bepaalde berekening meer van deze bewerkingen voorkomen, dan is de prioriteit als volgt:

1. eerst alle ↑;
2. dan alle * en /, die t.o.v. elkaar dezelfde prioriteit hebben en van links naar rechts worden uitgevoerd;
3. dan alle + en −, die eveneens t.o.v. elkaar dezelfde prioriteit hebben en van links naar rechts worden uitgevoerd.

Van deze prioriteit-regel kan worden afgeweken m.b.v. haakjes.
Enkele voorbeelden:

- 3 ↑ 2 + 4 ↑ 2 = 9 + 16 = 25
- 5 * 3 + 4 = 15 + 4 = 19
- (6 / 2 * 5 + 5) / 4 = (15 + 5) / 4 = 20 / 4 = 5
- 3 ↑ (2 + 3) * 2 = 3 ↑ 5 * 2 = 243 * 2 = 486

8. Print

De PRINT-statement kunnen we gebruiken voor:
1. het afdrukken van de tekst die achter PRINT tussen aanhalingstekens staat;
2. het afdrukken van het resultaat van een rekenkundige bewerking of van de waarde van de variable die achter PRINT staat;
3. het genereren van een extra LINE FEED-commando (= regel overslaan) als we helemaal niets achter PRINT aangeven.

Aan de hand van een voorbeeld zullen we deze drie mogelijkheden verduidelijken.

```
NEW
10 PRINT "TEKST 1"
20 PRINT "TEKST 2"
30 PRINT
40 PRINT (2 * 3 + 12) /9
50 END
RUN
TEKST 1
TEKST 2

    2
```

We merken het volgende op:
- Aan het begin van het programma hebben we het woord NEW ingetypt en op de RETURN-toets gedrukt. Hierdoor zijn alle statements uit een vorig programma die zich nog in het geheugen bevonden gewist.
- Nadat we het programma hebben ingetypt en afgesloten met END, hebben we het woord RUN ingetypt en daarna op de RETURN-toets gedrukt. Hierdoor is de vertaling en uitvoering van het programma gestart.
- De PRINT-statements op de regels 10 en 20 hebben tot gevolg dat de tekst TEKST 1 resp. TEKST 2, die tussen aanhalingstekens achter de PRINT-statements staan, op papier worden afgedrukt, en wel tegen de kantlijn.
- Achter de PRINT-statement op regel 30 staat niets aangegeven. Deze statement genereert dus alleen maar een LINEFEED-commando, zodat er 1 regel wordt overgeslagen.
- De PRINT-statement op regel 40 zorgt er voor, dat het resultaat van (2 × 3

+ 12) /9, dat is 2, op papier wordt afgedrukt. Omdat de computer voor het afdrukken van elk getal net zo veel print-posities reserveert als nodig is voor het afdrukken van het grootste getal dat kan worden verwerkt, wordt het cijfer 2 niet tegen de kantlijn afgedrukt.

Hoeveel print-posities de computer reserveert is afhankelijk van het type. Meestal zijn het er 11 of 14.

Uiteraard is het ook mogelijk om (2 × 3 + 12)/ 9 als tekst te interpreteren. We moeten deze bewerking dan tussen aanhalingstekens plaatsen:

```
NEW
10 PRINT "(2 * 3 = 12) / 9 ="
20 PRINT (2 * 3 + 12) / 9
30 END
RUN
(2 * 3 + 12) / 9 =
          2
```

Ook nu blijkt, dat bij de uitvoering van een *nieuwe* PRINT-statement op een *nieuwe* regel wordt begonnen, d.w.z. er wordt automatisch een RETURN- en een LINE FEED-commando gegenereerd. Willen we in bovenstaand voorbeeld het resultaat (2) achter de opdracht afdrukken, dan zullen we dus het RETURN- en LINE FEED-commando moeten onderdrukken. Dit gebeurt als we aan het eind van de PRINT statement een komma intypen.

```
NEW
10 PRINT "(2 * 3 + 12) / 9 =",
20 PRINT (2  3 + 12) / 9
30 END
RUN
(2 * 3 + 12) / 9 =          2
```

Deze komma kunnen we ook als scheidingsteken opvatten. D.w.z. we kunnen achter één PRINT-statement meerdere teksten of berekeningen aangeven, als we deze scheiden door komma's.

Bovenstaand programma kunnen we dus vereenvoudigen tot:

```
NEW
10 PRINT "(2 * 3 + 12) / 9 = ", (2 * 3 + 12) / 9
20 END
RUN
(2 * 3 + 12) / 9 =          2
```

Behalve een komma, kan ook een punt-komma als scheidingsteken worden gebruikt. De verschillende PRINT-gegevens worden dan tegen elkaar afgedrukt. In bovenstaand programma bijvoorbeeld is achter de „tekst" (2 * 3 + 12) / 9 het getal 2 afgedrukt. Daarbij is rekening gehouden met het grootste getal dat de computer kan weergeven en dat bijv. 11 of 14 print-posities in beslag neemt. Het getal 2 neemt echter slechts 1 print-positie in beslag, zodat zich tussen (2 * 3 + 12) / 9 en 2 een aantal spaties bevindt. Willen we dat nu niet, dan kunnen we i.p.v. de komma als scheidingsteken een punt-komma gebruiken.
Het programma wordt dan:

```
NEW
10 PRINT "(2 * 3 + 12) / 9 = "; (2 * 3 + 12) / 9
20 END
RUN
(2 * 3 + 12) / 9 =  2
```

Let op! Vóór een positief getal bevindt zich *altijd* 1 lege print-positie. Deze print-positie wordt bij een negatief getal nl. in beslag genomen door het −teken.

9. LET

De LET-statement dient om een waarde of de uitkomst van een expressie (d.i. een berekening) toe te kennen aan een variabele.
Een *variabele* is een grootheid die van waarde kan veranderen. In de computer beslaat een variabele één of meer geheugenlokaties. Deze geheugenlokaties kunnen we dus vullen met de LET-statement.
De *naam van een variabele* (ofte wel het adres van de geheugenlokaties) geven we aan met:

- één letter, of
- één letter, gevolgd door één cijfer.

Voorbeelden van juiste variabele-namen zijn:
X, A, Z, B9, D0, Y, E8, enz.
Voorbeelden van onjuiste variabele-namen zijn:
V26, 3X, IEEE, PIET, AB, R789, 18F, enz.
Slechts bij enkele computers mag de naam van een variabele uit meerdere letters bestaan. Raadpleeg hiervoor het handboek.
In de volgende 4 programmaregels worden waarden toegekend aan variabelen:

```
10 LET B = 10
20 LET X3 = 1736
30 LET D0 = -23
40 LET A = 0
```

M.b.v. de LET-statement is het ook mogelijk om de computer eerst een berekening uit te laten voeren, om dan het resultaat toe te kennen aan een variabele. Enkele voorbeelden hiervan zijn:

```
10 LET V7 = 83 + 6 -20
20 LET A = 20 * 4/8
30 LET Z = 1000 -(3*90 + 12)/3
```

In bovenstaande voorbeelden zijn achter het =teken alleen getallen vermeld. Het is echter ook mogelijk om hier variabele-namen te vermelden. De computer gebruikt voor de berekening dan de waarde die de variabele op dat moment heeft:

```
10 LET A = 50
20 LET B = 200
30 LET C = B/A
```

In dit voorbeeld krijgen A en B in regel 10 en 20 resp. de waarden 50 en 200. In regel 30 wordt eerst de berekening B/A uitgevoerd. Omdat A = 50 en B = 200, is B/A = 4. C krijgt in regel 30 dus de waarde 4.
Het is zelfs mogelijk om achter het =teken in de LET-statement dezelfde variabele aan te geven als vóór het =teken. Deze mogelijkheid hebben we bijv. nodig als we de waarde van een variabele met 1 willen verhogen.
Dit gebeurt dan als volgt:

```
10 LET A = A + 1
```

Allereerst wordt de berekening achter het =teken uitgevoerd, d.w.z. bij de waarde die A heeft, wordt 1 opgeteld.
Het resultaat wordt toegekend aan de variabele A, ofte wel de waarde van A is met 1 verhoogd.

Zoals we reeds weten, kunnen we met de PRINT-statement de waarde van een variabele op het beeldscherm (of met een printer) zichtbaar maken:

```
NEW
10 LET A = 50
20 LET B = 200
30 LET C = B/A
40 PRINT A, B, C
50 END
RUN
      50      200      4
```

Omdat in dit voorbeeld A, B en C in dezelfde PRINT-statement zijn vermeld (gescheiden door komma's) worden de getallen 50, 200 en 4 op dezelfde regel weergegeven; de RETURN- en LINE FEED-commando's worden door de komma's onderdrukt.

10. Berekening van de vervangingsweerstand

We gaan de computer de vervangingsweerstand laten berekenen voor een parallelschakeling van 2 weerstanden. De formule luidt:

$$Rv = \frac{R1 \cdot R2}{R1 + R2}$$

Deze formule moet eerst worden omgezet in een BASIC-expressie, omdat Rv geen geldige variabele-naam is en omdat het vermenigvuldig- en deelteken resp. * en / moet zijn.
Als we Rv vervangen door R0, wordt de formule:

R0 = R1 * R2 / (R1 + R2)

Stel dat de parallelschakeling bestaat uit weerstanden van 100 Ω en 400 Ω. Deze waarden moeten dan worden toegekend aan de variabelen R1 en R2:

```
NEW
10 LET R1 = 100
20 LET R2 = 400
```

Het resultaat van de berekening R1 * R2 / (R1 + R2) moet worden toegekend aan de variabele R0:

```
30 LET R0 = R1 * R2 / (R1 + R2)
```

en zichtbaar worden gemaakt op het beeldscherm:

```
40 PRINT R0
50 END
```

Het totale programma wordt dus:

```
NEW
10 LET R1 = 100
20 LET R2 = 400
30 LET R0 = R1 * R2 / (R1 + R2)
40 PRINT R0
50 END
RUN
      80
```

Uiteraard is het mogelijk om een en ander te verfraaien met wat tekst (afb. 2). In hoofdstuk 8 hebben we gezien dat we de af te drukken tekst dan tussen aanhalingstekens achter PRINT moeten plaatsen.

```
10 LET R1=100
20 LET R2=400
30 LET R0=R1*R2/(R1+R2)
40 PRINT"DE VERVANGINGSWEERSTAND VAN"
50 PRINT"TWEE WEERSTANDEN VAN";R1;"OHM"
60 PRINT"EN";R2;"OHM IS";R0;"OHM"
70 END
RUN
DE VERVANGINGSWEERSTAND VAN
TWEE WEERSTANDEN VAN 100 OHM
EN 400 OHM IS 80 OHM
```

Afb. 2

Bij de meeste interpreters mag in de LET-statement het woord LET worden weggelaten. Dit bespaart ons vaak veel typewerk tijdens het invoeren van het programma.

```
10 LET A = 50
20 LET B = 200
30 LET C = B/A
```

kunnen we dus vereenvoudigen tot:

```
10 A = 50
20 B = 200
30 C = B/A
```

11. INPUT

Zoals de PRINT-statement informatie uitvoert naar het uitvoer-apparaat (beeldscherm of printer), zo haalt de INPUT-statement informatie binnen vanaf het invoerapparaat, nl. het toetsenbord. Met de INPUT-statement kunnen we *tijdens de uitvoering van het programma* een waarde toekennen aan een variabele. Met de LET-statement doen we dat *tijdens het intypen van het programma.*
U zult zelf ongetwijfeld al de beperking hebben gezien van het programma van afb. 2. Dit programma kon nl. alleen maar de vervangingsweerstand berekenen voor een parallelschakeling van 2 weerstanden van 100 Ω en 400 Ω. Willen we deze waarden veranderen, dan moeten we het programma veranderen, nl. de regels 10 en 20.
Een veel betere oplossing is dan ook om i.p.v. de LET-statements in regel 10 en 20 INPUT-statements te gebruiken. Het programma komt er dan als volgt uit te zien:

```
NEW
10 INPUT R1
20 INPUT R2
30 LET R0 = R1 * R2 / (R1 + R2)
40 PRINT R0
50 END
```

Wanneer we nu de uitvoering van het programma starten door RUN in te typen (en daarna op de RETURN-toets te drukken) wordt t.g.v. de INPUT-statement op regel 10 een vraagteken op het beeldscherm afgedrukt en wacht de computer totdat wij een getal intypen op het toetsenbord en daarna op de RETURN-toets drukken.

```
RUN
? 25
```

Nadat wij het getal 25 hebben ingetypt en op de RETURN-toets hebben gedrukt, wordt dit getal toegekend aan de variabele R1, die in regel 10 achter het woord INPUT is aangegeven. De statement op regel 10 is nu uitgevoerd en de computer gaat verder bij regel 20. Dit is weer een INPUT-statement, zodat weer een vraagteken wordt afgedrukt:

```
RUN
? 25
? 100
       20
```

Achter dit 2e vraagteken hebben we nu het getal 100 ingetypt en daarna op de RETURN-toets gedrukt. De waarde 100 wordt toegekend aan de variabele R2, die achter INPUT in regel 20 is aangegeven.
Nu is ook de statement op regel 20 uitgevoerd en komt het programma bij regel 30 waar R0 wordt berekend. In regel 40 tenslotte, wordt de waarde van R0, dit is 20, afgedrukt.

Het voordeel van het gebruik van de INPUT-statement boven dat van de LET-statement is dus, dat één en hetzelfde programma geldt voor willekeurige waarden van R1 en R2.
Omdat we aan een variabele alleen een getal kunnen toekennen, mogen we achter het vraagteken, dat t.g.v. een INPUT-statement verschijnt, *alleen cijfers intypen, en dus geen letters.* In het laatste geval zal de computer nl. een foutmelding geven en de statement opnieuw uitvoeren, d.w.z. er verschijnt opnieuw een vraagteken.
Hoe vreemd het ook mag klinken, maar met de INPUT-statement kan ook tekst worden weergegeven, en wel op dezelfde manier als bij de PRINT-statement door het tussen aanhalingstekens in de INPUT-regel op te nemen.

```
NEW
10 INPUT "DE WAARDE VAN A IS", A
20 INPUT "DE WAARDE VAN B IS", B
30 PRINT "A + B = ", A + B
40 END
RUN
DE WAARDE VAN A IS ? 12
DE WAARDE VAN B IS ? 9
A + B =  21
```

U ziet dat eerst de tekst wordt afgedrukt en dat daarna het "INPUT-vraagteken" verschijnt.

Aan het opnemen van tekst in een INPUT-statement zijn wel enkele voorwaarden verbonden, nl.:

1. Er kan slechts *eenmaal* tekst worden opgenomen. De statement
   10 INPUT "DE WAARDE VAN A IS", A, "HALLO"
   is ongeldig en zal een foutmelding opleveren.

2. De INPUT-statement kan niet dienen om alleen tekst weer te geven. Er moet ook een variabele in voorkomen die via het toetsenbord een waarde krijgt. De statement

   50 INPUT "PROGRAMMEREN IN BASIC"

   bevat geen variabele en is dus ongeldig.
3. De tekst die we willen weergeven moet vóór de variabele(n) staan.

   300 INPUT A, "IS DE WAARDE VAN A"

   is eveneens een ongeldige statement en veroorzaakt een foutmelding.

12. De weergave van getallen

In de vorige hoofdstukken hebben we alleen gewerkt met gehele getallen, zgn. *integers*. Bij de meeste interpreters moeten deze getallen uit minder dan 7 cijfers bestaan.
Een computer die alleen met deze getallen kan werken, is natuurlijk vrij beperkt in zijn mogelijkheden. Bij de programmeertaal BASIC is dan ook voorzien in het werken met decimale breuken, zgn. *reals*.

Een decimale breuk die door ons via het toetsenbord wordt ingevoerd of die het resultaat is van een berekening, wordt door de computer in 2 delen weergegeven, nl.:

- een getal tussen −1 en +1, dit is de *mantisse;*
- een macht van 10, de *exponent*.

Enkele voorbeelden zijn:

.4657838E 02 = 0,4657838 × 10^2 = 46,57838
.9876812E 00 = 0,9876812 × 10^0 = 0,9876812
.2647198E 12 = 0,2647198 × 10^{12} = 264719800000
.7563491E−03 = 0,7563491 × 10^{-3} = 0,0007563491

Er gelden de volgende regels:

1. I.p.v. de decimale komma, zoals we die gewend zijn, wordt een punt gebruikt.
2. De mantisse bestaat, afhankelijk van het type computer, meestal uit 6 of 9 cijfers.
3. De 0 voor de decimale punt wordt niet weergegeven. Achter de letter E staat alleen de exponent.
5. Het laatste cijfer van de mantisse is het resultaat van een afronding.
6. De exponent van de macht van 10 mag min. −99 en max. +99 zijn. Het grootste getal dat kan worden weergegeven is dus .9999999E 99 (Ook dit kan van computer tot computer verschillen. Veel voorkomende max. en min. waarden voor de exponent zijn resp. +37 en −38. Raadpleeg hiervoor het handboek).
7. Een getal dat uit meer dan 6 cijfers bestaat, en dat door de computer moet worden verwerkt, wordt ook als decimale breuk weergegeven.

Eén en ander zullen we nu verduidelijken aan de hand van een programmavoorbeeld.

```
NEW
10 LET A = 0.25 + 0.39
20 LET B = 0.034* 3
30 LET C = A + B
40 PRINT A, B, C
50 INPUT Z
60 PRINT Z
70 END
RUN
 .6400000E 00  .1020000E 00  .7420000E 00
?43.648
 .4364800E 02
```

In regel 10 krijgt A de waarde 0.25 + 0.39 = 0.64. Deze waarde wordt weergegeven als .6400000E 00. Zoals u ziet mogen we bij het intypen van decimale breuken wel de 0 voor de decimale punt vermelden; het hoeft echter niet. Een decimale komma mag echter nooit worden gebruikt.
B krijgt in regel 20 de waarde 0.034*3 = 0.102, hetgeen wordt weergegeven als .1020000E 00. In regel 30 worden A en B bij elkaar opgeteld. Het resultaat, 0.742, wordt toegekend aan de variabele C en weergegeven als .7420000E 00.
Nadat deze 3 waarden t.g.v. de PRINT-statement op regel 40 zijn afgedrukt, volgt in regel 50 een INPUT-statement. Deze statement heeft tot gevolg, dat een vraagteken wordt afgedrukt en dat de computer wacht totdat er via het toetsenbord een getal wordt ingetypt.
In bovenstaand voorbeeld hebben we het getal 43.648 ingevoerd. U ziet dat dit getal t.g.v. de PRINT-statement op regel 60 wordt weergegeven als
.4364800E 02.

Indien de laatste cijfers van de mantisse 0 zijn, worden deze bij veel computers weggelaten. Het getal .7420000E 00 bijvoorbeeld, wordt dan weergegeven als .742E 00.

13. Goto

Alle programma's die we tot nu toe hebben behandeld zijn in oplopende regelnummervolgorde uitgevoerd; we begonnen steeds bij regel 10 en eindigden bij regel 50 of 60.
M.a.w. wanneer het programma bij de laatste regel was aangekomen, was het beëindigd. Wilden we het programma nogmaals uitvoeren, dan moest opnieuw RUN worden ingetypt. U zult begrijpen dat dit niet zo handig is en er is dan ook de mogelijkheid om van deze „uitvoering in oplopende regelnummervolgorde" af te wijken.
We kunnen dit doen met een zgn. *sprong-opdracht*, nl. de GOTO („ga naar")-statement. Achter GOTO vermelden we het nummer van die regel waar naar toe moet worden gesprongen. Let op! GOTO wordt aan elkaar geschreven.
Laten we eens een voorbeeld behandelen van een programma dat twee door ons in te typen getallen bij elkaar optelt en het resultaat op het beeldscherm weergeeft. We zullen het programma eerst schrijven zonder gebruik te maken van de GOTO-statement.

```
NEW
10 INPUT A
20 INPUT B
30 PRINT "DE SOM IS",
40 PRINT A + B
50 END
RUN
?16
?33
DE SOM IS      49
```

De programma-uitvoering is nu beëindigd. Willen we het programma nogmaals uitvoeren, dan moeten we opnieuw RUN intypen:

```
RUN
?41
?15.763
DE SOM IS      .5676300E 02
```

Nu gaan we gebruik maken van de GOTO-statement. Aan het einde van het

programma, d.w.z. na de PRINT-statement op regel 40, voegen we de volgende statement tussen:

```
45 GOTO 10
```

Aangekomen bij regel 45 krijgt de computer dus de opdracht om in het programma terug te springen naar regel 10, d.w.z. naar het begin.
Voor de zekerheid vragen we even een listing op van hetgeen zich nu in het geheugen bevindt:

```
LIST
10 INPUT A
20 INPUT B
30 PRINT "DE SOM IS",
40 PRINT A + B
45 GOTO 10
50 END
```

Nu hoeven we het programma slechts eenmaal te starten; het wordt steeds opnieuw uitgevoerd:

```
RUN
?67
?19
DE SOM IS              86
?9111
?6
DE SOM IS              9117
?24.876543
?0
DE SOM IS   .2487654E 02
?
```

Vragen:

1. Wat is de functie van de komma aan het einde van regel 30?
2. Is de END-statement op regel 50 nog wel nodig?
3. Bovenstaand programma is in feite een „oneindige lus''. Steeds wanneer de computer aankomt bij regel 45, volgt een sprong terug naar regel 10. Hoe kunnen we nu tóch de uitvoering van dit programma onderbreken?
4. Wat gebeurt er als we achter een vraagteken dat t.g.v. één van de INPUT-statements verschijnt bijv. ABCDE intypen?

Antwoorden:

1. De komma onderdrukt het RETURN- en LINE FEED-commando, zodat het resultaat van de berekening A + B achter de tekst „DE SOM IS'' wordt afgedrukt.
2. Nee. Het programma springt vanaf regel 45 altijd naar regel 10.

3. Door tegelijkertijd op de toetsen CTRL en C te drukken of, bij andere typen computers, door op de toets BREAK of ESCAPE te drukken.
4. Aan de variabelen A en B kunnen we alleen getallen toekennen; geen tekst. De computer zal dan ook een foutmelding geven en de statement opnieuw uitvoeren, d.w.z. er verschijnt opnieuw een vraagteken.

14. IF-THEN

Stel, we willen een programma schrijven dat de getallen 1 t/m 5 op het beeldscherm weergeeft en daarna stopt. Laten we maar eens iets proberen...

```
NEW
10 LET A = 1
20 PRINT A
30 LET A = 2
40 PRINT A
50 LET A = 3
60 PRINT A
70 LET A = 4
80 PRINT A
90 LET A = 5
100 PRINT A
110 END
RUN
        1
        2
        3
        4
        5
```

Dit programma werkt uiteraard prima, maar u ziet zelf ook wel dat het veel korter kan. Immers, we hebben een variabele A die steeds met 1 wordt verhoogd en waarvan daarna steeds de waarde op het beeldscherm zichtbaar wordt gemaakt.
Het programma kan dus ook als volgt worden geschreven:

```
NEW
10 LET A = 0
20 LET A = A + 1
30 PRINT A
40 GOTO 20
50 END
```

ware het niet, dat nu de getallen 1 t/m „oneindig" worden afgedrukt omdat dit

programma nl. nooit ophoudt. Er wordt immers *altijd* teruggesprongen naar regel 20.
Wat we echter willen, is dat in regel 40 alleen naar regel 20 wordt teruggesprongen *als A kleiner of gelijk is aan 5;* we moeten immers alleen de getallen 1 t/m 5 op het beeldscherm weergeven. Gelukkig biedt BASIC ook voor dit probleem een oplossing. Er bestaat nl. een *voorwaardelijke sprong-opdracht.* Deze heeft de volgende vorm:

IF *voorwaarde* THEN *statement*

Achter IF moeten we een bepaalde voorwaarde vermelden, bijv. X = 9, G6>100, B = −1, enz.
Achter THEN moeten we een statement opgeven. Dit mag elke willekeurige statement zijn en dus ook een GOTO-statement.
Nu wilden we in bovenstaand voorbeeld alleen terugspringen naar regel 20 als A kleiner of gelijk was aan 5, ofte wel kleiner dan 6. De onvoorwaardelijke GOTO-statement in regel 40 gaan we daarom vervangen door:

40 IF A < 6 THEN GOTO 20

Wanneer nu op een gegeven moment niet meer aan de voorwaarde A < 6 wordt voldaan, wordt het gedeelte achter THEN niet meer uitgevoerd (er wordt dus niet meer teruggesprongen) en wordt *verdergegaan met de volgende statement.* Dit is in ons voorbeeld de statement op regel 50 zodat het programma stopt.
De zgn. *operators* die we in de voorwaarde achter IF mogen gebruiken, zijn:

= gelijk aan
< > ongelijk aan
> groter dan
< kleiner dan
> = groter dan of gelijk aan
< = kleiner dan of gelijk aan

In bovenstaand programma hadden we dus ook de volgende statement kunnen gebruiken:

40 IF A < = 5 THEN GOTO 20

Sommige BASIC-interpreters accepteren de opdracht "THEN GOTO" niet. U moet dan òf alleen THEN vermelden, òf alleen GOTO. Raadpleeg hiervoor het handboek of probeer het even uit.
Regel 40 in bovenstaand programma wordt dan:

40 IF A < 6 THEN 20

òf:

40 IF A < 6 GOTO 20

In sommige gevallen, zoals bij de PET-computer, kunnen beide mogelijkheden worden gebruikt.

Bij veel BASIC-interpreters kunnen in één IF-statement meerdere voorwaarden worden aangegeven. Er wordt daarbij gebruik gemaakt van zgn. *„logical operators"*. De bekendste zijn AND (en) en OR (of).

100 IF A > 10 AND B = 6 THEN PRINT X

Bij deze statement wordt de opdracht PRINT X *alleen dan* uitgevoerd wanneer èn A groter is dan 10 èn B gelijk is aan 6.

150 IF A < 0 OR A > 10 OR P = 20 THEN 100

T.g.v. deze statement wordt naar regel 100 gesprongen wanneer aan minstens één van de drie voorwaarden (A < 0, A > 10 of P = 20) is voldaan. Is aan geen van de voorwaarden voldaan, dan wordt verder gegaan met de statement die volgt op regel 150.

200 IF K < 5 OR (L > 10 AND Z = 20) THEN 300

Hier hebben we eigenlijk te maken met twee voorwaarden:

1. K < 5
2. L > 10 èn Z = 10

Alleen wanneer aan één van deze beide voorwaarden of aan beide is voldaan, wordt naar regel 300 gesprongen.

15. Stroomdiagrammen

De programma's die in de vorige hoofdstukken als voorbeeld zijn behandeld, werden allemaal „uit de losse hand" opgesteld. We moesten een bepaald probleem oplossen m.b.v. een programma en begonnen dan maar direct met het opschrijven van de statements. Dit was mogelijk, omdat het allemaal korte „recht toe-recht aan" programma's waren.
Anders wordt het, wanneer we te maken krijgen met wat langere programma's waarin ook nog eens diverse sprongen voorkomen. In zo'n geval doen we er

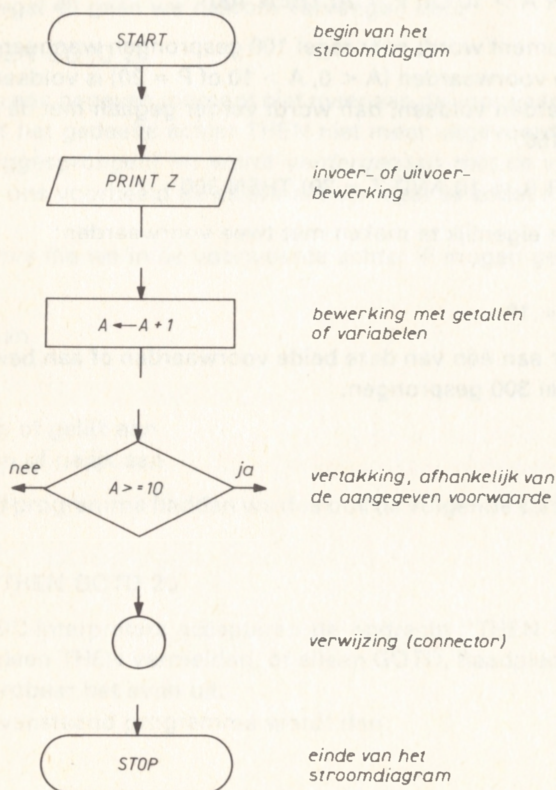


Fig. 3

goed aan (en meestal komen we er niet eens onderuit) om een zgn. stroomdiagram of stroomschema op te stellen.

Met behulp van een stroomdiagram kunnen we de oplossingsmethode voor een probleem schematisch beschrijven. De oplossingsmethode zelf, moet natuurlijk wel eerst in ons hoofd aanwezig zijn.

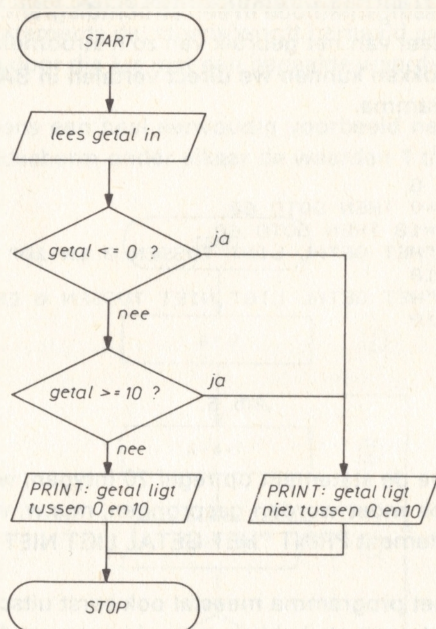


Fig. 4

In fig. 3 is weergegeven welke figuren we ter beschikking hebben om een stroomdiagram mee op te bouwen. Deze blokken worden, om tot het eigenlijke stroomdiagram te komen, verbonden door lijnen, waarin m.b.v. een pijl is aangegeven in welke richting het diagram wordt doorlopen.

Voorbeeld

We gaan een BASIC-programma schrijven, dat aangeeft of een getal, dat we via het toetsenbord intypen, *tussen* 0 en 10 ligt.

De oplossingsmethode is als volgt:
Eerst wordt een getal ingelezen in de computer. Dan wordt onderzocht of het getal kleiner dan of gelijk is aan 0, of dat het getal groter dan of gelijk is aan 10. Wordt aan één van deze voorwaarden voldaan, dan moet de computer de tekst „het getal ligt niet tussen 0 en 10" afdrukken. Wordt niet aan deze voorwaarde voldaan, dan drukt de computer de tekst „het getal ligt tussen 0 en 10" af. Daarna moet worden teruggesprongen naar het begin van het programma, zodat een volgend getal kan worden onderzocht.
Fig. 4 toont deze oplossingsmethode in een stroomdiagram. U ziet nu hopelijk direct het grote voordeel van het gebruik van zo'n stroomdiagram: de teksten in de verschillende blokken kunnen we direct vertalen in BASIC-statements.
Afb. 5 toont het programma.

```
10 INPUT G
20 IF G<=0 THEN GOTO 60
30 IF G>=10 THEN GOTO 60
40 PRINT"HET GETAL LIGT TUSSEN 0 EN 10"
50 GOTO 10
60 PRINT"HET GETAL LIGT NIET TUSSEN 0 EN 10"
70 GOTO 10
80 END
```

Afb. 5

Op het moment dat we de statement op regel 20 intypen, weten we uiteraard nog niet waar naar toe moet worden gesprongen, m.a.w. we weten nog niet op welke regel de statement PRINT "HET GETAL LIGT NIET TUSSEN 0 en 10" komt te staan.
Daarom moeten we het programma meestal ook eerst uitschrijven op papier. Pas wanneer we alle statements hebben staan, kunnen we de sprongadressen (de regelnummers waar naar toe moet worden gesprongen) invullen.

16. FOR-NEXT

Het komt erg vaak voor, dat in een programma een lus (Engels: loop) voorkomt, dus een aantal statements dat steeds wordt herhaald en waarbij een variabele bij elke rondgang door die lus met een bepaalde waarde wordt verhoogd, bijv. met 1.

Laten we maar eens een heel eenvoudig voorbeeld nemen. Een programma moet op het beeldscherm onder elkaar de waarden 1 t/m 10 weergeven.

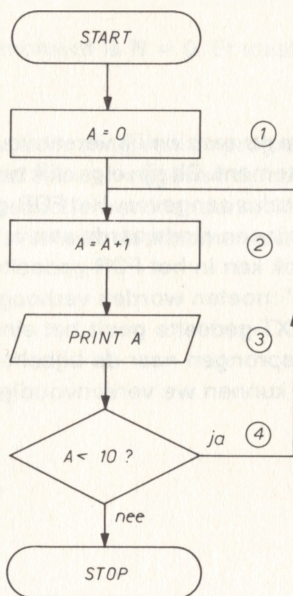


Fig. 6

Een stroomdiagram, dat de oplossingsmethode voor dit probleem beschrijft, is weergegeven in fig. 6. We zien een programmalus, gevormd door de blokken 2, 3 en 4, waarin de variabele A steeds met 1 wordt verhoogd en wordt weergegeven totdat A de waarde 10 heeft. Aan het begin dient A natuurlijk te worden gedefinieerd (A = 0).

Het BASIC-programma ziet er als volgt uit:

```
NEW
10 LET A = 0
20 LET A = A + 1
30 PRINT A
40 IF A < 10 THEN GOTO 20
50 END
RUN
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
```

Dit programma kunnen we nu aanzienlijk vereenvoudigen door gebruik te maken van de FOR-NEXT-statement. Dit zijn eigenlijk twee statements die het *begin en eind van een programmalus* aangeven. Het FOR-gedeelte staat aan het begin van de lus en geeft de begin- en eindwaarde aan van de variabele die in de lus van waarde verandert. Ook kan in het FOR-gedeelte worden aangegeven met hoeveel de „lusvariabele" moeten worden verhoogd of verlaagd (dit laatste is nl. ook mogelijk). Het NEXT-gedeelte geeft het einde van de lus aan en zorgt ervoor dat wordt teruggesprongen naar de bijbehorende FOR-statement.
Bovenstaand programma kunnen we vereenvoudigen tot:

```
NEW
10 FOR A = 1 TO 10
20 PRINT A
30 NEXT A
40 END
```

Wat er nu gebeurt is het volgende. In regel 10 krijgt A de waarde 1 en in regel 20 wordt deze waarde op het beeldscherm weergegeven. In regel 30 wordt A met 1 verhoogd en wordt onderzocht of de eindwaarde die in het FOR-gedeelte was aangegeven is *overschreden*. Wanneer dit nog niet het geval is, wordt teruggesprongen naar de FOR-statement waarin de variabele die achter NEXT is aangegeven was gedefinieerd. Wanneer we weer aankomen bij regel 10 heeft A dus de waarde 2 gekregen (dit is in regel 30 gebeurd). In regel 10 gebeurt nu niets en er wordt verdergegaan bij regel 20: de waarde 2 wordt op het beeldscherm weergegeven. Wanneer de lus op deze manier 10 maal is doorlopen ko-

men we weer aan bij regel 30 waar de variabele A, die inmiddels de waarde 10 heeft, met 1 wordt verhoogd. Hierdoor overschrijdt A de eindwaarde die in de FOR-statement als 10 was gedefinieerd. *A wordt dus wel 11, maar er wordt niet teruggesprongen naar regel 10.*

Samengevat komt het er op neer, dat het FOR-gedeelte de begin- en eindwaarde bepaalt van de lusvariabele, maar dat de waardeverandering van deze variabele in het NEXT-gedeelte plaatsvindt. Wanneer nà deze waardeverandering blijkt, dat de in het FOR-gedeelte aangegeven eindwaarde is overschreden, wordt niet meer teruggesprongen naar deze FOR-statement en wordt verdergegaan met de statement die volgt op NEXT.

Een veel voorkomende fout is de volgende:
In een programma komt de statement

100 FOR A = 1 TO N

voor, en op een gegeven moment is N = 0. Er staat dan:

100 FOR A = 1 TO 0

Het is dan waarschijnlijk de bedoeling van de programmeur dat de statements binnen de FOR-NEXT lus niet worden uitgevoerd, maar dat gebeurt nu juist wel. Immers, pas in het NEXT-gedeelte wordt onderzocht of de eindwaarde is overschreden, maar dan is het al te laat: de statements binnen de lus zijn dan al één keer uitgevoerd.

17. STEP

In bovenstaand voorbeeld is in de FOR-statement alleen de begin- en eindwaarde vermeld; er is niet aangegeven wat de waardeverandering van de variabele bij elke rondgang door de lus moest zijn. Nu is het zo, dat wanneer niets wordt aangegeven de waardeverandering + 1 is; in het vorige hoofdstuk is A steeds met 1 verhoogd.
Willen we een andere waardeverandering aangeven, dan kan dit m.b.v. het woord STEP dat we achter de FOR-statement intypen.
Enkele voorbeelden:

```
NEW
10 FOR X = 8 TO 16 STEP 2
20 PRINT X
30 NEXT X
40 END
RUN
       8
      11
      12
      14
      16
```

```
NEW
10 FOR T = 5 TO 20 STEP 4
20 PRINT T
30 NEXT T
40 END
RUN
       5
       9
      13
      17
```

```
NEW
10 FOR D = 10 TO 1 STEP -2
20 PRINT D
30 NEXT D
40 END
RUN
      10
       8
       6
       4
       2
```

De begin- en eindwaarde die in de FOR-statement worden aangegeven, hoeven niet per se getallen te zijn. We kunnen ook variabelen vermelden, waarvan de waarden bijv. via het toetsenbord kunnen worden ingevoerd:

```
NEW
 10 PRINT "BEGINWAARDE=",
 20 INPUT B
 30 PRINT "EINDWAARDE=",
 40 INPUT E
 50 PRINT "STAPGROOTTE=",
 60 INPUT S
 70 FOR G=B TO E STEP S
 80 PRINT G
 90 NEXT G
100 END
RUN
BEGINWAARDE = ?5
EINDWAARDE = ?30
STAPGROOTTE = ?8
          5
         13
         21
         29
```

In dit programma krijgt B in regel 20 de waarde 5 die we via het toetsenbord intypen. Op dezelfde manier krijgen E en S in de regels 40 en 60 resp. de waarden 30 en 8.
De computer vervangt de statement op regel 70, dus door

```
70 FOR G = 5 TO 30 STEP 8
```

Bij de eerste rondgang door de programmalus, gevormd door de regels 70, 80 en 90, heeft G dan de waarde 5. Elke NEXT-statement op regel 90 verhoogt G steeds met 8.
Wanneer het programma voor de 4de maal bij regel 90 aankomt, wordt G weer verhoogd. G was 29 en wordt 29 + 8 = 37. Hierdoor wordt de eindwaarde overschreden en wordt verdergegaan met de statement die volgt op NEXT. Dit is de END-statement zodat het programma stopt.

Uiteraard kan het voorkomen dat er in één programma meerdere FOR-NEXT lussen voorkomen. Een belangrijke regel is in dit geval, dat deze lussen elkaar niet mogen kruisen.
In fig. 7 zijn enkele voorbeelden gegeven van een *geldige* combinatie van FOR-NEXT lussen. In geen van de gevallen doorkruisen de lussen elkaar. Vanuit een NEXT-statement wordt automatisch teruggesprongen naar de bijbehorende

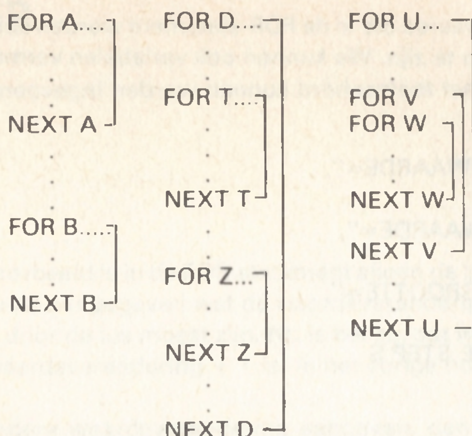


Fig. 7

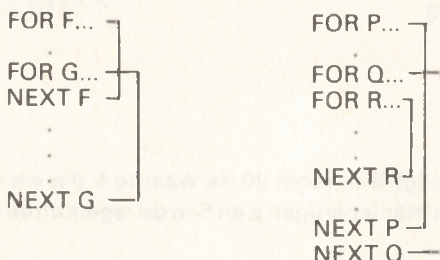


Fig. 8

FOR-statement, d.w.z. de FOR-statement met dezelfde vaiabele.
Fig. 8 toont 2 voorbeelden van elkaar kruisende en dus *ongeldige* FOR-NEXT lussen.

Voorbeeld
In afb. 9 is een programma weergegeven waarin twee FOR-NEXT lussen voorkomen.
In regel 10 wordt de variabele K gedefinieerd. De beginwaarde van K is 10; de eindwaarde is 13. Omdat in deze FOR-statement geen STEP is aangegeven, zal K bij elke rondgang door de lus met 1 worden verhoogd.
In regel 20 wordt de variabele L gedefinieerd. De beginwaarde van L is 100; de eindwaarde is 200. Na elke rondgang door de lus zal L met 25 worden verhoogd (STEP 25).

Wat gebeurt er nu bij de uitvoering van dit programma?
Bij aankomst op regel 30 heeft K de waarde 10 en L de waarde 100. Deze waarden worden t.g.v. de PRINT-statement op het beeldscherm weergegeven. Dan komen we bij regel 40, waar L met 25 wordt verhoogd. Hierdoor wordt de eindwaarde niet overschreden, zodat wordt teruggesprongen naar regel 20. Daar gebeurt in feite niets en de PRINT-statement op regel 30 geeft de waarden 10 en 125 weer op het beeldscherm.

```
10 FOR K=10 TO 13
20 FOR L=100 TO 200 STEP 25
30 PRINT K,L
40 NEXT L
50 PRINT
60 NEXT K
70 END
RUN

    10          100
    10          125
    10          150
    10          175
    10          200

    11          100
    11          125
    11          150
    11          175
    11          200

    12          100
    12          125
    12          150
    12          175
    12          200

    13          100
    13          125
    13          150
    13          175
    13          200
```

Afb. 9

In regel 40 wordt L weer met 25 verhoogd en er wordt weer teruggesprongen naar regel 20. Dit gaat zo door totdat L de eindwaarde 200 heeft overschreden. Dan wordt nl. na regel 40 verdergegaan bij regel 50. Hier staat een „loze" PRINT-opdracht die er voor zorgt dat 1 regel extra wordt overgeslagen. Dan komen we aan bij regel 60 waar de variabele K met 1 wordt verhoogd. K was nog steeds 10 en wordt 11. Er wordt nu teruggesprongen naar regel 10 waar verder niets gebeurt.
In regel 20, en nu komt het, *wordt L opnieuw gedefinieerd.* L wordt dus opnieuw 100 en zal na elke rondgang door de lus met 25 worden verhoogd.

Vragen

1. Wat wordt ten gevolge van onderstaand programma op het beeldscherm weergegeven?

```
NEW
10 FOR A = 1 TO 5
20 PRINT A
30 NEXT A
40 PRINT A
50 END
RUN
```

2. Wat voor uitwerking hebben de volgende statements?

```
10 PRINT,
20 END
```

3. Wat wordt t.g.v. onderstaand programma op het beeldscherm weergegeven?

```
NEW
10 FOR T = -1 TO 1
20 PRINT T,
30 NEXT T
40 END
```

Antwoorden

1. De regels 10, 20 en 30 hebben tot gevolg, dat de getallen 1, 2, 3, 4 en 5 worden weergegeven. Wanneer de NEXT-statement op regel 30 echter voor de 5e maal wordt uitgevoerd, wordt A met 1 verhoogd. A was 5 en wordt dus 6.
   Hierdoor wordt de in regel 10 gestelde eindwaarde overschreden zodat de computer verdergaat met de statement die volgt op NEXT. Dit is de statement PRINT A, zodat de waarde 6 op het beeldscherm wordt weergegeven.
2. Een „loze" PRINT-statement genereert een extra RETURN/LINE-FEED-commando. De komma achter het woord PRINT onderdrukt deze echter weer, zodat het totale resultaat van deze statement nihil is.
3. De beginwaarde van T is −1; de eindwaarde is +1. Achter de FOR-statement is geen STEP aangegeven, zodat T steeds met 1 wordt verhoogd. De waarden − 1, 0 en +1 worden dus op het beeldscherm weergegeven en omdat achter de regel 20 een komma staat, gebeurt dit achter elkaar.

18. Master Mind

In de voorgaande hoofdstukken hebben we ongeveer de helft van alle BASIC-statements behandeld, nl. PRINT, LET, INPUT, GOTO, IF-THEN en FOR-NEXT-STEP.
Om te laten zien wat met deze paar statements reeds mogelijk is en om aan te tonen hoe we ze moeten combineren, geven we in dit hoofdstuk een soort samenvatting. We gaan een programma schrijven voor het spel „Master Mind".

De spelregels zijn als volgt. De computer genereert vier willekeurige getallen tussen 0 en 10 (d.w.z. min. 1 en max. 9) en slaat die op in zijn geheugen. De tegenspeler krijgt die getallen niet te zien, maar zal proberen om ze te raden. Hij typt daartoe op het toetsenbord vier getallen in, waarna de computer deze getallen vergelijkt met de getallen die hij zelf heeft gegenereerd. Dan „vertelt" de computer de tegenspeler, via het beeldscherm of de printer, hoeveel getallen goed zijn geraden en hoeveel er op de goede plaats staan.
Aan de hand van een voorbeeld zullen we dit verduidelijken. Stel, de computer genereert de volgende getallen:

3 7 4 8

(bij het feitelijke spel krijgen we die dus niet te zien).
De tegenspeler typt nu in:

1 7 8 4

De computer zal ons nu de volgende melding moeten geven:

„3 getallen goed, waarvan 1 op de goede plaats".

Wanneer we dit een aantal malen herhalen, kunnen we aan de hand van de antwoorden van de computer concluderen, welke getallen de computer heeft gegenereerd en in welke volgorde die getallen staan. Wanneer we de getallen hebben geraden, meldt de computer dat en is het spel beëindigd.

Analyse
We zullen nu de opdracht gaan analyseren, zodat we aan de hand hiervan een stroomdiagram kunnen opstellen dat kan worden vertaald in een programma.

De computer krijgt te maken met twee reeksen getallen, zgn. *getallenrijen*. Laten

we deze A en B noemen. Rij A bestaat uit de getallen die de computer zelf genereert; rij B bevat de getallen die via het toetsenbord worden ingetypt.
De programmeertaal BASIC kent voor het weergeven van dergelijke getallenrijen speciale variabele-namen, zgn. *subscripted variabels* (variabelen met een index). De variabele-namen worden gevormd door een letter (plus eventueel een cijfer), gevolgd door een getal tussen haakjes. De letter duidt op de naam van de getallenrij, terwijl het getal tussen haakjes de plaats van de afzonderlijke getallen (elementen) in die rij aangeeft.
Voorbeelden van juiste „subscripted variabels" zijn: A(3), K(1), Z(76), G3(5), B(2), enz.

Uiteraard mogen we het getal tussen de haakjes ook d.m.v. een „gewone" variabele aangeven: een variabele vertegenwoordigt immers een getal. Met A(N) bijv. bedoelen we het N^e getal uit rij A.

Wat moet de computer dus doen?

1. Een getallenrij A genereren die bestaat uit willekeurige elementen (getallen). Deze elementen worden aangegeven met A(1), A(2), A(3) en A(4).
2. Om het spel niet al te moeilijk te maken, moeten dit verschillende getallen zijn. Nadat rij A is gegenereerd moet de computer daarom onderzoeken of de rij getallen bevat die gelijk zijn. Is dit het geval, dan wordt een nieuwe rij gegenereerd.
3. Nu worden vier getallen ingelezen van het toetsenbord. Er wordt onderzocht of deze getallen binnen het vereiste bereik liggen (1 t/m 9) en of ze alle verschillend zijn. De getallen worden toegekend aan getallenrij B en heten B(1), B(2), B(3) en B(4).
4. Alle elementen uit rij A worden vergeleken met alle elementen uit rij B. Wanneer de computer een gelijkheid aantreft, wordt een variabele G met 1 verhoogd (aan het begin van het programma wordt G op nul ingesteld).
   Wanneer de computer een gelijkheid aantreft terwijl ook de nummers van de elementen uit rij A en rij B gelijk zijn (bijv. A(2) en B(2)), dan wordt tevens een variabele P met 1 verhoogd. P geeft dan aan hoeveel van de ingetypte getallen „op de goede plaats staan". Aan het begin van het programma wordt ook P op nul gesteld.
5. Wanneer P gelijk is aan vier, betekent dit dat we de getallen die de computer had gegenereerd hebben geraden. Het spel is dan afgelopen en de computer meldt dit.
6. Is P kleiner dan vier, dan meldt de computer: „G getallen goed, waarvan P op de goede plaats".
   Hierbij worden uiteraard de getalwaarden van P en G weergegeven.
7. De tegenspeler moet nu de mogelijkheid hebben om nogmaals te raden, m.a.w. er moet worden „teruggesprongen naar punt 3".

Stroomdiagrammen
We hebben nu de oplossingsmethode opgesteld en kunnen die gaan omzetten in een zgn. *algemeen stroomdiagram*. In dit stroomdiagram (fig. 10) is de oplossingsmethode heel globaal weergegeven.

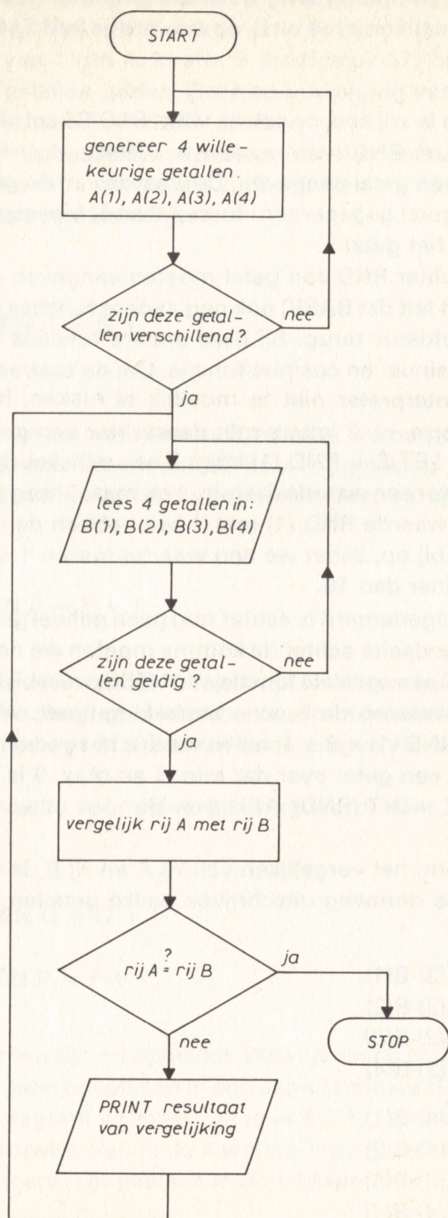


Fig. 10

De blokken van dit stroomdiagram moeten nu worden opgesplitst in een aantal kleinere blokken, zodat een zgn. gedetailleerd stroomdiagram ontstaat. Voordat we dit kunnen doen, moeten er nog twee problemen worden opgelost:

- hoe genereert de computer een „willekeurig" getal?
- hoe worden de getallenrijen A en B, op een beetje handige manier, met elkaar vergeleken?

Het eerste probleem is vrij snel opgelost, want BASIC kent hiervoor een speciale zgn. BASIC-functie, *nl.* RND (van random = willekeurig). Achter RND moeten we tussen haakjes een getal aangeven. De waarde van dit getal is onbelangrijk; er wordt altijd een getal gegenereerd tussen 0 en 1. Meestal vermelden we dan ook tussen haakjes het getal 1.
De reden dat we achter RND een getal moeten aangeven dat op zich niet belangrijk is, ligt in het feit dat BASIC ook nog andere functies kent. Hierop komen we in een later hoofdstuk terug. Bij deze andere functies is dat getal wel belangrijk, bijv. bij de sinus- en cosinus-functie. Om de zaak eenduidig te houden, *d.w.z.* om het de interpreter niet te moeilijk te maken, hebben alle BASIC-functies dezelfde vorm, nl. 3 letters met daarachter een getal tussen haakjes. *T.g.v.* de statement LET Z = RND (1) krijgt Z een willekeurige waarde *tussen* 0 en 1. We wilden echter een waarde die min. 1 en max. 9 mag zijn. Daarom vermenigvuldigen we de waarde RND (1) met 9 (we hebben dan een getal *tussen* 0 en 9) en tellen er 1 bij op, zodat we een waarde *tussen* 1 en 10 krijgen, d.w.z. groter dan 1 en kleiner dan 10.
Het getal dat nu is gegenereerd is echter nog geen geheel getal, maar een breuk, bijv. 6,345186. Het gedeelte achter de komma moeten we nog weghalen en ook hiervoor kent BASIC een speciale functie, nl. INT(X), waarbij X het zgn. argument is, d.w.z. het getal waarop de functie betrekking heeft. Wanneer we de INT-functie nemen van RND (1) × 9 + 1, halen we dus het gedeelte achter de komma weg en houden we een getal over dat min. 1 en max. 9 is.
Samengevat: LET Z = INT (RND) (1) × 9 + 1).

Het tweede probleem, het vergelijken van rij A en rij B, is iets ingewikkelder. Laten we maar eens domweg uitschrijven welke getallen met elkaar moeten worden vergeleken.

A(1)-B(1)	A(3)-B(1)
A(1)-B(2)	A(3)-B(2)
A(1)-B(3)	*A(3)-B(3)*
A(1)-B(4)	A(3)-B(4)
A(2)-B(1)	A(4)-B(1)
A(2)-B(2)	A(4)-B(2)
A(2)-B(3)	A(4)-B(3)
A(2)-B(4)	*A(4)-B(4)*

De vergelijkingen A(1)-B(1), A(2)-B(2), A(3)-B(3) en A(4)-B(4) zijn cursief weergegeven, omdat de computer hier, bij het aantreffen van een gelijkheid, niet alleen de variabele G, maar ook de variabele P met 1 moet verhogen.
In bovenstaande tabel zien we dat het elementnummer van rij A achtereenvolgens 1, 2, 3 en 4 is. Voor elk elementnummer in rij A loopt het elementnummer van rij B ook van 1 t/m 4. In afb. 9 (hoofdstuk 17) hebben we eenzelfde opeenvolging van getallen gezien. Deze opeenvolging van getallen werd toen verwezenlijkt m.b.v. twee FOR-NEXT-lussen.
Dezelfde truc kunnen we ook nu toepassen. Het elementnummer van rij A noemen we N; het elementnummer van rij B noemen we M. De vergelijking van rij A en rij B verloopt dan als volgt:

```
FOR N = 1 TO 4
FOR M = 1 TO 4
IF A(N) = B(M) THEN G = G + 1
NEXT M
NEXT N
```

Voor het bepalen van de waarde van P, moeten we nu nog de vergelijkingen A(1)-B(1), A(2)-B(2), enz. uitvoeren. Bij deze vergelijking kunnen we voor de elementnummers dezelfde variabele gebruiken. Deze variabele moet lopen van 1 t/m 4:

```
FOR Z = 1 TO 4
IF A(Z) = B(Z) THEN P = P + 1
NEXT Z
```

Deze beide programma's kunnen natuurlijk heel eenvoudig worden gecombineerd. In het eerste programma hebben we immers al een variabele die (eenmaal) van 1 t/m 4 loopt, nl. N. Deze variabele mogen we best nog eens gebruiken.
Het programmagedeelte voor de vergelijking wordt dan:

```
FOR N = 1 TO 4
FOR M = 1 TO 4
IF A(N) = B(M) THEN G = G + 1
NEXT M
IF A(N) = B(N) THEN P = P + 1
NEXT N
```

De grootste problemen zijn nu opgelost. We kunnen de blokken van het algemene stroomdiagram gaan opsplitsen in een aantal kleinere blokken, zodat een gedetailleerd stroomdiagram ontstaat. Dit is in fig. 11, 12 en 13 weergegeven.
Fig. 11 toont het gedeelte waarin de 4 willekeurige gehele getallen tussen 0 en 10 worden gegenereerd. Dit gebeurt in de blokken 2 t/m 5.

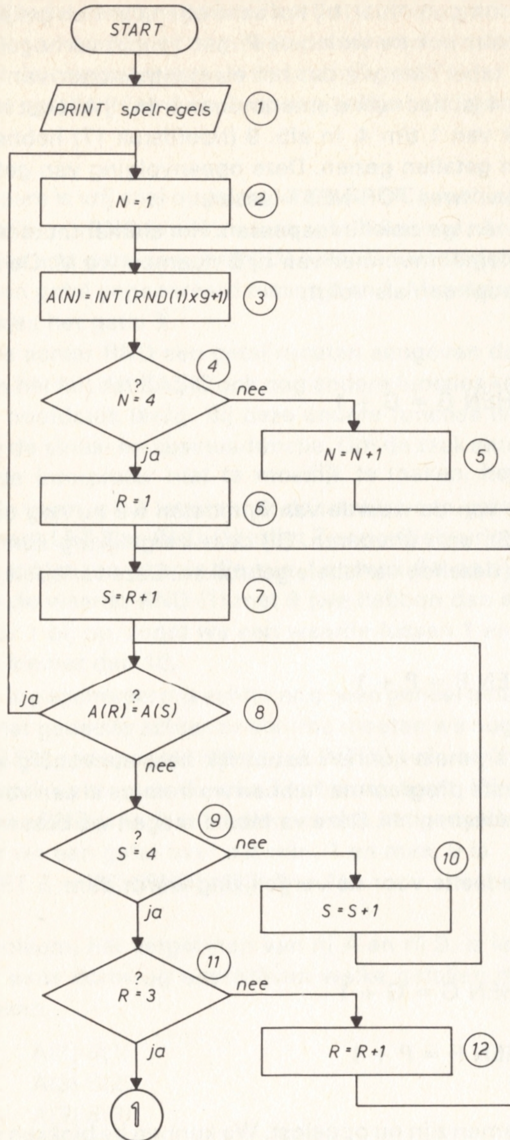

START
PRINT spelregels
1
N = 1
2
A(N) = INT(RND(1)x9+1)
3
4
N ? = 4
nee
N = N+1
5
ja
R = 1
6
S = R+1
7
ja
A(R) ? = A(S)
8
nee
9
S ? = 4
nee
10
S = S+1
ja
11
R ? = 3
nee
R = R+1
12
ja
1

Fig. 11

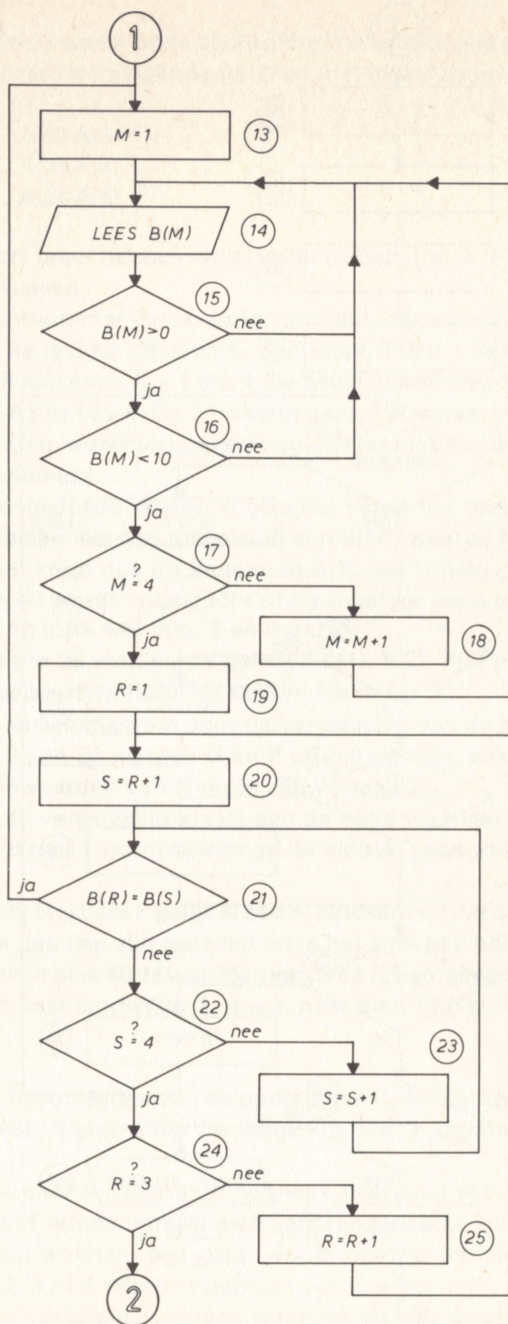
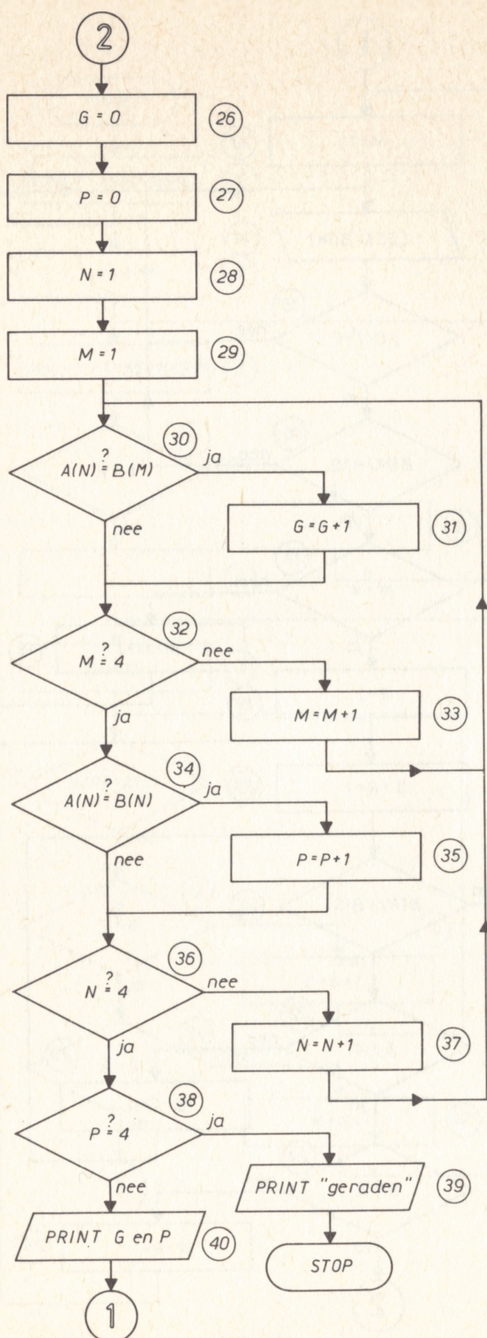

1
M = 1
13
LEES B(M)
14
B(M)>0
15
nee
ja
B(M)<10
16
nee
ja
M ? 4
17
nee
ja
M = M + 1
18
R = 1
19
S = R + 1
20
ja
B(R) = B(S)
21
nee
S ? 4
22
nee
ja
S = S + 1
23
R ? 3
24
nee
ja
R = R + 1
25
2

Fig. 12



2
G = 0
26
P = 0
27
N = 1
28
M = 1
29
30
A(N) ?= B(M)
ja
nee
G = G + 1
31
32
M ?= 4
nee
ja
M = M + 1
33
34
A(N) ?= B(N)
ja
nee
P = P + 1
35
36
N ?= 4
nee
ja
N = N + 1
37
38
P ?= 4
ja
nee
PRINT "geraden"
39
STOP
PRINT G en P
40
1

Fig. 13

In de blokken 6 t/m 12 wordt onderzocht of het 4 verschillende getallen zijn. Voor dit onderzoek moeten de volgende getallen met elkaar worden vergeleken:

A(1)-A(2)	A(2)-A(3)
A(1)-A(3)	A(2)-A(4)
A(1)-A(4)	A(3)-A(4)

Een dergelijk soort tabel hebben we al vaker gezien. Het is te realiseren m.b.v. twee FOR-NEXT-lussen.
Als we het elementnummer van de linker getallen in bovenstaande tabel R noemen en dat van de rechter getallen S, dan loopt R van 1 t/m 3, terwijl S voor elke waarde van R loopt van R + 1 t/m 4. De beide lussen waarin S en R variëren zijn in de blokken 6 t/m 12 van fig. 11 weergegeven. Wanneer in blok 8 een gelijkheid wordt gevonden, wordt teruggesprongen naar blok 2 zodat 4 nieuwe getallen worden gegenereerd.
Fig. 12 toont het gedeelte waarin 4 getallen vanaf het toetsenbord worden ingelezen. De getallen worden toegekend aan B(M), waarbij M loopt van 1 t/m 4. Het eerste getal krijgt dus de benaming B(1), het laatste getal heet B(4). In de blokken 15 en 16 wordt onderzocht of elk ingetypt getal resp. groter is dan 0 en kleiner dan 10 (ofte wel min. 1 en max. 9).
In de blokken 19 t/m 25 worden de getallen B(1), B(2), B(3) en B(4) met elkaar vergeleken. Dit gebeurt op dezelfde manier als in fig. 11.
Fig. 13 toont het stroomdiagram voor de vergelijking van de getallenrijen A en B. In de blokken 26 en 27 worden G en P op nul gesteld, terwijl in de blokken 28 en 29 de beginwaarden van N en M zijn vermeld.
In blok 30 vindt de vergelijking plaats aan de hand waarvan G, bij het vinden van een gelijkheid met 1 wordt verhoogd. In blok 34 gebeurt hetzelfde voor de variabele P.
Wanneer in blok 39 blijkt dat P gelijk is aan 4, betekent dit dat de door de computer gegenereerde getallen zijn geraden en is het spel beëindigd. Is dit niet het geval, dan worden in blok 40 de waarden van G en P weergegeven waarna wordt teruggesprongen naar connector 1, d.w.z. naar blok 13 (fig. 12).

Programma
De laatste stap is het omzetten van de opdrachten in de blokken van het stroomdiagram naar BASIC-statements. We krijgen dan het programma dat in afb. 14 is weergegeven.
Dit programma kunnen we uiteraard nog verfraaien met een hoeveelheid tekst die we m.b.t. PRINT-statements kunnen aanbrengen, bijv. voor het weergeven van foutmeldingen wanneer een getal wordt ingetypt dat niet tussen 0 en 10 ligt, of wanneer 2, 3 of 4 dezelfde getallen worden ingetypt, enz.
Ook kan bijv. een variabele T, steeds wanneer de vier getallen worden ingevoerd, met 1 worden verhoogd, zodat T aan het einde van het spel aangeeft in

hoeveel beurten de vier getallen zijn geraden. Afhankelijk van de waarde van T kan dan een waardering worden afgedrukt, zoals amateur, beginneling, opvallend goed, zeker vaker gedaan, enz. ... In afb. 15 is een voorbeeld gegeven. Er zijn uiteraard nog vele andere mogelijkheden, die we aan uw eigen initiatief en creativiteit willen overlaten.

```
10 PRINT"*******MASTER MIND*******"
20 PRINT
30 PRINT"PROBEER 4 VERSCHILLENDE GETALLEN TE RADEN TUSSEN 0 EN 10"
40 FOR N=1 TO 4
50 A(N)=INT(RND(1)*9+1)
60 NEXT N
70 FOR R=1 TO 3
80 FOR S=R+1 TO 4
90 IF A(R)=A(S) THEN GOTO 40
100 NEXT S
110 NEXT R
120 FOR M=1 TO 4
130 INPUT'B(M)
140 IF B(M)<1 THEN GOTO 130
150 IF B(M)>9 THEN GOTO 130
160 NEXT M
170 FOR R=1 TO 3
180 FOR S=R+1 TO 4
190 IF B(R)=B(S) THEN GOTO 120
200 NEXT S
210 NEXT R
220 G=0
230 P=0
240 FOR N=1 TO 4
250 FOR M=1 TO 4
260 IF A(N)=B(M) THEN G=G+1
270 NEXT M
280 IF A(N)=B(N) THEN P=P+1
290 NEXT N
300 IF P=4 THEN GOTO 500
310 PRINT G;"GETAL(LEN) GOED, WAARVAN";P;"OP DE GOEDE PLAATS"
320 GOTO 120
500 PRINT"GERADEN"
510 END
RUN
 *******MASTER MIND*******
 PROBEER 4 VERSCHILLENDE GETALLEN TE RADEN TUSSEN 0 EN 10
 ?3
 ?4
 ?8
 ?1
 2 GETAL(LEN) GOED, WAARVAN 0 OP DE GOEDE PLAATS
```

Afb. 14

Opmerking:
Op sommige computers (bijv. de Wang 2200) zal dit programma niet werken. Dit komt omdat de lengten van de getallenrijen niet zijn gedefineerd. We komen hier in het volgende hoofdstuk op terug.

```
10 PRINT"*******MASTER MIND*******"
20 PRINT
30 PRINT"PROBEER 4 VERSCHILLENDE GETALLEN TE RADEN TUSSEN 0 EN 10"
35 T=0
40 FOR N=1 TO 4
50 A(N)=INT(RND(1)*9+1)
60 NEXT N
70 FOR R=1 TO 3
80 FOR S=R+1 TO 4
90 IF A(R)=A(S) THEN GOTO 40
100 NEXT S
110 NEXT R
120 FOR M=1 TO 4
130 INPUT B(M)
140 IF B(M)<1 THEN GOTO 700
150 IF B(M)>9 THEN GOTO 700
160 NEXT M
170 FOR R=1 TO 3
180 FOR S=R+1 TO 4
190 IF B(R)=B(S) THEN GOTO 600
200 NEXT S
210 NEXT R
215 T=T+1
220 G=0
230 P=0
240 FOR N=1 TO 4
250 FOR M=1 TO 4
260 IF A(N)=B(M) THEN G=G+1
270 NEXT M
280 IF A(N)=B(N) THEN P=P+1
290 NEXT N
300 IF P=4 THEN GOTO 500
310 PRINT G;"GETAL(LEN) GOED, WAARVAN";P;"OP DE GOEDE PLAATS"
320 GOTO 120
500 PRINT"GERADEN IN";T;"BEURTEN"
510 IF T<4 THEN GOTO 550
520 IF T<8 THEN GOTO 560
530 IF T<12 THEN GOTO 570
540 PRINT"WAARDELOOS"
545 END
550 PRINT"OPVALLEND GOED"
555 GOTO 545
560 PRINT"AMATEUR"
565 GOTO 545
570 PRINT"BEGINNELING"
575 GOTO 545
600 PRINT"IK ZEI 4 VERSCHILLENDE GETALLEN"
610 PRINT"PROBEER HET NOG EENS"
620 GOTO 120
700 PRINT"HET GETAL MOET TUSSEN 0 EN 10 LIGGEN"
710 GOTO 130
```

Afb. 15

Mocht het programma niet werken, neem dan de volgende statement op:
5 DIM A(4), B(4)

Vragen (bij afb. .15):

1. Mag regel 50 ook worden geschreven als:

   50 A(N) = INT (RND (1)*9) + 1

(let op het laatste haakje).
2. Vervang regel 140 en regel 150 door één statement.
3. Wanneer wordt de statement op regel 540 uitgevoerd?
4. Wordt het intypen van ongeldige waarden (< 1, > 9 of gelijke getallen) als één beurt geteld?

Antwoorden:

1. Ja. Het getal 1 is een geheel getal. INT (1) = 1 en het getal 1 mag dus ook achteraf bij INT (RND (1) 9) worden opgeteld.
2. 140 IF B(M) < 1 OR B(M) > 9 THEN GOTO 700
3. Als T groter dan of gelijk aan 12.
4. Nee. De „teller" T wordt pas verhoogd als alle waarden voor B(M) geldig zijn, d.w.z. in regel 215.

19. DIM

In het vorige hoofdstuk hebben we reeds kennis gemaakt met getallenrijen. Een getallenrij bestaat uit een aantal, bij elkaar behorende getallen, ook wel elementen genoemd. Voor het verwerken van getallenrijen kent de programmeertaal BASIC speciale variabele-namen, zgn. *„subscripted variabels"* (geïndiceerde variabelen). Deze variabele-namen bestaan uit een letter, eventueel gevolgd door een cijfer, met daar achter een getal tussen haakjes. De letter en het eventuele cijfer geven de naam van de getallenrij aan; het getal tussen haakjes geeft aan welk element uit die rij wordt bedoeld. G(7) bijvoorbeeld is het 7e element uit rij G.

Wanneer we in een BASIC-programma van dergelijke getallenrijen gebruik maken, worden ze opgeslagen in het geheugen van de computer. Er moet dus een aantal geheugenplaatsen voor worden gereserveerd.
Nu is het bij de meeste computers niet per se noodzakelijk dat wij aangeven hoeveel geheugenplaatsen voor elke rij moeten worden gereserveerd; de computer kan dit nl. ook zelf, maar reserveert dan voor elke rij die hij tegenkomt een *vast aantal plaatsen,* meestal te veel, zodat onnodig veel geheugenruimte verloren gaat die anders voor de opslag van het programma zelf kon worden gebruikt. Daarom doen we er goed aan om aan het begin van een programma aan te geven, hoe lang de in het programma voorkomende getallenrijen zijn.
We doen dit m.b.v. de DIM-statement (DIM van DIMension = omvang, grootte). Achter het woord DIM geven we de naam van de rij aan, en daarachter tussen haakjes uit hoeveel elementen die rij bestaat.

Voorbeeld:

10 DIM A(12)

M.b.v. deze statement reserveren we 12 geheugenplaatsen voor de elementen van rij A.
Uiteraard houdt dit in, dat rij A uit niet meer dan 12 elementen mag bestaan (minder mag natuurlijk wel).

Met één DIM-statement is het mogelijk om geheugenruimte te reserveren voor meerdere rijen:

10 DIM A(12), H(46), Z(5)

Met deze statement wordt ruimte gereserveerd voor rij A (max. 12 elementen), rij H (max. 46 elementen) en rij Z (max. 5 elementen).

Behalve getallenrijen, kunnen we de computer ook *matrices* laten verwerken. Een matrix is een aantal bij elkaar horende getallen die zijn *gerangschikt in rijen en kolommen*. We kunnen eigenlijk ook zeggen: een matrix bestaat uit een aantal getallenrijen.
Stel, dat we bijvoorbeeld een aantal temperatuurwaarden willen verwerken die, gedurende 5 dagen, elke 4 uur zijn gemeten. We krijgen dan een volgende tabel (matrix) met in elk hokje een gemeten waarde.

	0.00 uur	4.00 uur	8.00 uur	12.00 uur	16.00 uur	20.00 uur
1e dag	12	11	13	22	20	18
2e dag	13	10	10	17	17	16
3e dag	10	10	14	23	22	20
4e dag	15	14	16	27	26	22
5e dag	15	13	14	25	22	19

Deze matrix bestaat uit 5 rijen en 6 kolommen. Om de gemeten waarden in het geheugen van de computer op te slaan, doen we er goed aan de juiste hoeveelheid geheugenruimte te reserveren met een DIM-statement:

```
10 DIM T(5,6)
```

Dit betekent: reserveer ruimte voor een matrix T, bestaande uit 30 elementen ingedeeld in 5 rijen en 6 kolommen.
In het volgende hoofdstuk zullen we laten zien hoe we de waarden uit de tabel in de computer kunnen invoeren.

Opmerking:
In het voorgaande zijn we er van uitgegaan dat het eerste element van een getallenrij elementnummer 1 heeft. Echter, elementnummer 0 mag ook worden gebruikt.
Het eerste element heeft dan nummer 0, het tweede element heeft nummer 1, enz. Met DIM A(6) reserveren we in feite 7 geheugenlocaties voor rij A met de elementen 0 t/m 6.
U merkt wel, dat deze wijze van nummering nogal verwarrend is en ze wordt dan ook zelden gebruikt, tenzij het gaat om een zo efficiënt mogelijk gebruik van de geheugenruimte.

20. READ en DATA

In voorgaande hoofdstukken hebben we de LET- en INPUT-statements behandeld. Beide dienen om een waarde (getal) toe te kennen aan een variabele. Bij de LET-statement is die waarde tijdens het schrijven van het programma reeds bekend en wordt ze vermeld in de LET-statement. Bij de INPUT-statement is de waarde afkomstig van het toetsenbord tijdens de uitvoering van het programma.
Wanneer we aan een aantal variabelen waarden moeten toekennen, of wanneer aan één variabele achtereenvolgens een aantal waarden moet worden toegekend, kunnen we dit uiteraard doen met LET-statements. Dit neemt echter een groot aantal programmaregels en dus geheugenruimte in beslag. Een betere methode is dan ook, om gebruik te maken van de READ- en DATA-statement.
De *READ-opdracht* wordt gebruikt om, tijdens de uitvoering van het programma een waarde toe kennen die in een *DATA-regel* is opgenomen.
De DATA-regels kunnen in principe op elke willekeurige plaats in het programma worden opgenomen. Om het programma overzichtelijk te houden, plaatsen we ze echter bij voorkeur helemaal aan het eind.
(Bij sommige BASIC-interpreters zijn voor de DATA-regels speciale regelnummers gereserveerd, bijv. 5000 en hoger. In zo'n geval hoeven we het woord „DATA" niet te vermelden, maar kunnen we de getallen, gescheiden door komma's, direct achter de regelnummers vermelden).
In een DATA-regel kunnen meerdere waarden worden vermeld. Bij een *volgende READ-opdracht* wordt automatisch een *volgende waarde* uit de DATA-regel ingelezen.

Voorbeeld:

```
NEW
10 READ Z
20 PRINT "Z=";Z
30 READ Y
40 PRINT "Y=";Y
50 DATA 6,28
60 END
RUN
Z= 6
Y= 28
```

Dit programma laat zien dat de eerste READ-opdracht tot gevolg heeft dat de eerste DATA-waarde wordt ingelezen; Z krijgt de waarde 6. De tweede READ-opdracht op regel 30 heeft tot gevolg dat de tweede DATA-waarde wordt ingelezen. Deze waarde wordt toegekend aan de variabele Y.
Dit programma kan nog worden verkort door het lezen van de DATA-waarden m.b.v. één READ-opdracht uit te voeren. De variabelen moeten dan worden gescheiden door een komma:

```
NEW
10 READ Z, Y
20 PRINT "Z=", Z
30 PRINT "Y=",Y
40 DATA 6,28
60 END
```

We zullen nu laten zien hoe de waarden worden toegekend aan de elementen van een getallenrij. De rij noemen we R, het elementnummer N.

```
10 DIM R(10)
20 FOR N=1 TO 10
30 READ R(N)
40 NEXT N
50 DATA 45,6,-26,1,7889
60 DATA 4,6,0,-457,2
70 END
```

In regel 10 wordt geheugenruimte gereserveerd voor max. 10 elementen van rij R.
De regels 20, 30 en 40 worden elk 10 maal doorlopen, waarbij steeds een volgende DATA-waarde wordt ingelezen. Deze waarden zijn in dit voorbeeld verdeeld over twee regels. Wanneer alle waarden uit regel 50 zijn ingelezen, wordt automatisch verdergegaan bij regel 60.
Het kan voorkomen dat in de DATA-regel(s) minder getallen zijn opgenomen, dan er READ-opdrachten zijn. In dat geval wordt een foutmelding gegeven wanneer een READ-opdracht wordt uitgevoerd terwijl er geen DATA-waarden meer aanwezig zijn.
Is het aantal DATA-waarden groter dan het aantal READ-opdrachten, dan worden de overtollige waarden eenvoudigweg niet gebruikt.

Zoals in het vorige hoofdstuk beloofd, zouden we een programma schrijven om een matrix van 6 kolommen bij 5 rijen te vullen met een aantal meetwaarden. Welnu, dat is eigenlijk heel eenvoudig nu we de READ-DATA-opdracht kennen (vandaar dat we het even hebben uitgesteld). De matrix heet T, het kolomnummer noemen we K en het rijnummer R.
We gaan de matrix rij voor rij vullen. R wordt 1 en K loopt op van 1 tot en met

6. Dan wordt R gelijk aan 2 en loopt K weer op van 1 tot en met 6, enz.
U voelt het al aan: twee FOR-NEXT-lussen in elkaar, de buitenste met de variabele R en de binnenste met de variabele K. De eigenlijke lus-inhoud bestaat maar uit 1 statement, nl. READ T(R,K), en in een aantal DATA-regels vermelden we, rij voor rij, de gemeten waarden.
Het programma ziet er dus als volgt uit.

```
NEW
 10 DIM T(5,6)
 20 FOR R=1 TO 5
 30 FOR K=1 TO 6
 40 READ T(R,K)
 50 NEXT K
 60 NEXT R
 70 DATA 12, 11, 13, 22, 20, 18
 80 DATA 13, 10, 10, 17, 17, 16
 90 DATA 10, 10, 14, 23, 22, 20
100 DATA 15, 14, 16, 27, 26, 22
110 DATA 15, 13, 14, 25, 22, 19
120 END
```

21. Bepaling van het gemiddelde

In het volgende voorbeeld gaan we een programma schrijven dat het gemiddelde bepaalt van een aantal in een DATA-regel opgenomen waarden.
Om het gemiddelde van een rij getallen te kunnen bepalen, moeten we eerst alle getallen optellen en daarna de som delen door het aantal getallen. We krijgen dus te maken met twee tellers: N geeft het aantal getallen aan en S de som van de getallen.
Fig. 16 toont het stroomdiagram. Allereerst worden de tellers N en S op nul gezet. Dan wordt de eerste waarde ingelezen en wordt N met 1 verhoogd. Bij S wordt de zojuist ingelezen waarden opgeteld.
Is N nog steeds kleiner dan 10, dan wordt de volgende waarde ingelezen en opgeteld bij S. N wordt weer met 1 verhoogd. Is N 10 geworden, dan zijn alle waarden aan de beurt geweest en kan het gemiddelde eenvoudig worden bepaald door S te delen door N.
Het programma ziet er als volgt uit:

```
10 LET N=0
20 LET S=0
30 READ G
40 LET N=N + 1
50 LET S=S + G
60 IF N < 10 THEN 30
70 PRINT S/N
80 DATA 45,3,99,0,6,1,-654,8,10,5
90 END
```

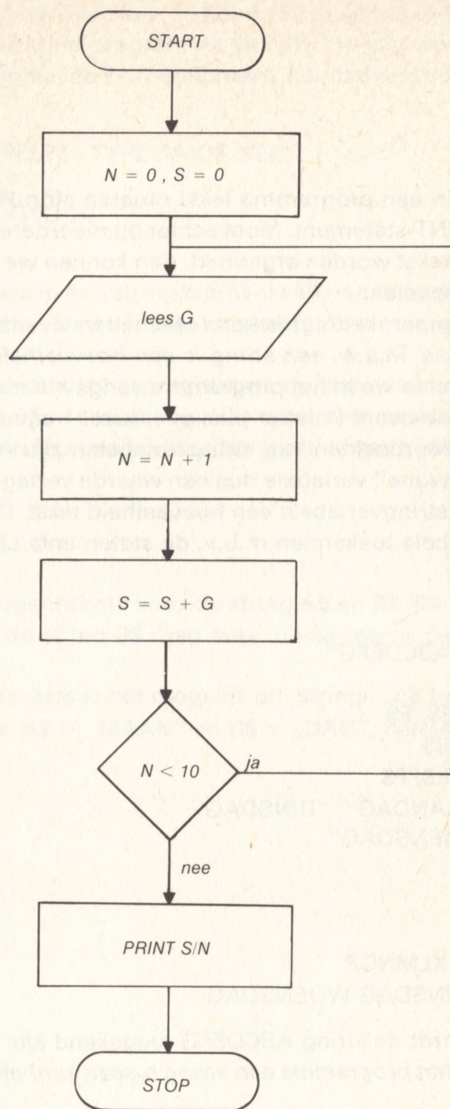


Fig. 16

22. Stringvariabelen

Wanneer we in een programma tekst moeten afdrukken kunnen we dat doen m.b.v. een PRINT-statement. Moet echter op meerdere plaatsen in het programma eenzelfde tekst worden afgedrukt, dan kunnen we beter gebruik maken van zgn. stringvariabelen.
Een *string* (=snoer, ketting) is een reeks letters eventueel gevolgd door cijfers en/of leestekens. M.a.w. *een string is een hoeveelheid tekst.*
Een string kunnen we in het programma aangeven met een *stringvariabele,* dit is een variabele-naam (1 letter plus eventueel 1 cijfer), gevolgd door een dollarteken ($). Voorbeelden van stringvariabelen zijn A$, F$, J$, B3$, Z9$, enz.
Zoals een „gewone" variabele dus een waarde vertegenwoordigt, zo vertegenwoordigt een stringvariabele een hoeveelheid tekst. Deze tekst kunnen we aan de stringvariabele toekennen m.b.v. de statements LET, INPUT en READ.

Voorbeeld:

```
10 LET A$= "ABCDEFG"
20 INPUT B$
30 READ D$, E$, F$
40 PRINT A$, B$
50 PRINT D$, E$, F$
60 DATA "MAANDAG", "DINSDAG"
70 DATA "WOENSDAG"
80 END
RUN
?HIJKLMNOP
ABCDEFG HIJKLMNOP
MAANDAG DINSDAG WOENSDAG
```

In regel 10 wordt de string ABCDEFG toegekend aan de stringvariabele A$. U ziet dat we *in het programma een string tussen aanhalingstekens* moeten plaatsen.
T.g.v. de INPUT-statement op regel 20 wordt een vraagteken afgedrukt. Hierachter hebben we HIJKLMNOP ingetypt (let op! nu niet tussen aanhalingstekens).
Deze string wordt toegekend aan de stringvariabele B$.
De strings die in de DATA-regels (regel 60 en 70) zijn opgenomen, worden toegekend aan de stringvariabelen D$, E$ en F$.

T.g.v. de PRINT-statements op regel 40 en 50 worden alle strings afgedrukt. Stringvariabelen kunnen ook worden gebruikt in combinatie met de IF-THEN-statement. We zullen dit laten zien aan de hand van onderstaand deelprogramma dat achter het Master Mind-programma van afb. 14 kan worden gevoegd. Regel 510 van dit programma, de END-statement, wordt overschreven door een nieuwe statement.

```
510 PRINT "NOG EEN SPEL??  TYPE JA OF NEE"
520 INPUT A$.
530 IF A$-"JA" THEN GOTO 20
540 END
```

Het maximale aantal tekens in een string is afhankelijk van het gebruikte computersysteem en de interpreter. De gewenste lengte kan worden opgegeven m.b.v. de DIM-opdracht.
Nemen we geen DIM-opdracht in het programma op, dan reserveert de computer zelf geheugenruimte voor elke string. Het maximale aantal tekens is dan meestal 15. Wordt dit aantal overschreden, dan geeft de computer een foutmelding.
Met de statement:

```
10 DIM A$(5),B$(58)
```

reserveren we dus geheugenruimte voor de string A$ en B$. De string A$ mag max. 5 tekens bevatten, de string B$ mag max. uit 58 tekens bestaan.

Opmerking: Bij vele computers is het mogelijk om strings „op te tellen", d.w.z. samen te voegen. Is bijv. A$ = „MAAN" en B$ = „DAG", dan is C$ = A$ + B$ = „MAANDAG"

23. Stringfuncties

In dit hoofdstuk behandelen we enkele stringfuncties die niet tot de standaard-BASIC horen, maar wel aanwezig zijn op de meeste personal computers, zoals de PET, TRS-80, enz.
M.b.v. deze stringfuncties kunnen bepaalde bewerkingen worden uitgevoerd met een string.

LEN-functie

Met de LEN-functie (LEN van Length) kan de lengte van een string, d.w.z. het aantal tekens, worden bepaald.
Voorbeeld:

```
10 LET A$ = "PROGRAMMEREN"
20 LET L = LEN(A$)
30 PRINT L
40 END
RUN
     12
```

In regel 20 wordt de stringfunctie LEN gebruikt. Het argument van deze functie, d.w.z. datgene waarop de functie betrekking heeft (in dit geval de stringvariabele A$) is tussen haakjes geplaatst. Dit is nl. bij elke BASIC-functie het geval. We zullen dit verderop vaker tegenkomen.
In regel 20 krijgt L de waarde LEN(A$), d.w.z. L wordt gelijk aan het aantal tekens in de string PROGRAMMEREN.
In regel 20 is het argument van de LEN-functie een stringvariabele. We kunnen echter ook een string vermelden. Voorbeeld:

```
10 LET L = LEN ("MICROCOMPUTER")
20 PRINT L
30 END
RUN
     13
```

Ook nu moet de string weer tussen aanhalingstekens worden geplaatst.

LEFT$-functie

Vaak is het gewenst om slechts *een deel van een string* aan te kunnen spreken. Dit is o.a. mogelijk met de functie LEFT$. M.b.v. deze functie kunnen we nl. de linker N tekens van een string toekennen aan een nieuwe stringvariabele.

```
10 DIM Z$(21)
20 LET Z$ = "PROGRAMMEREN IN BASIC"
30 LET V$ = LEFT$(Z$12',
40 PRINT V$
50 END
RUN
PROGRAMMEREN
```

In regel 10 wordt geheugenruimte gereserveerd voor de string Z$, bestaande uit 21 tekens. Let op! De beide spaties tussen de woorden tellen ook mee, de aanhalingstekens niet.
In regel 20 wordt de string PROGRAMMEREN IN BASIC toegekend aan de stringvariabele Z$. De functie LEFT$(Z$,12) neemt de linker 12 tekens van de string Z$ en kent ze toe aan de stringvariabele V$. In regel 40 wordt de string V$ afgedrukt.

RIGHT$-functie

Met deze functie kan het rechter deel van een string worden aangesproken. De schrijfwijze van deze functie is gelijk aan die van de LEFT$-functie.
Voorbeeld:

```
NEW

10 DIM Z$ (12)
20 LET Z$ = "PROGRAMMEREN"
30 FOR N = 1 TO LEN(Z$)
40 PRINT RIGHT$(Z$,N)
50 NEXT N
60 END
RUN
N
EN
REN
EREN
MEREN
MMEREN
AMMEREN
RAMMEREN
```

(vervolg op volgende pagina)

GRAMMEREN
OGRAMMEREN
ROGRAMMEREN
PROGRAMMEREN

Nadat in regel 10 geheugenruimte is gereserveerd voor de string Z$, wordt in regel 20 de string PROGRAMMEREN toegekend aan Z$.
In regel 30 is de functie LEN(Z$) opgenomen. Deze functie berekent de lengte van de string Z$. Regel 30 wordt door de computer a.h.w. vervangen door:

30 FOR N = 1 TO 12

De statement op regel 40 zorgt ervoor dat de rechter N tekens van de string Z$ worden afgedrukt. De eerste maal dat regel 40 wordt uitgevoerd is N = 1, de tweede maal is N = 2, de derde maal is N = 3, enz. De eindwaarde van N is 12. De eerste keer wordt dus alleen het meest rechtse teken afgedrukt. De tweede keer de 2 rechtse tekens, de derde keer de 3 rechtse tekens, enz.

MID$-functie

Met deze functie kunnen één of meer tekens die zich op een nader aan te geven plaats in een string bevinden worden aangesproken.
De algemene vorm van deze functie is:

MID$(stringvariabele, N, M)

Hierbij geeft N de plaats aan van het eerste teken dat men wil aanspreken en M geeft het aantal tekens aan.
Voorbeeld:

```
NEW
10 LET D$ = "PROGRAMMEREN"
20 LET E$ = MID$(D$,5,3)
30 PRINT E$
40 END
RUN
RAM
```

De functie MID$(D$,5,3) neemt 3 tekens uit de string D$, te beginnen bij het 5^e^ teken. De nieuwe string, bestaande uit de tekens R, A en M wordt toegekend aan de stringvariabel E$.

Nog een voorbeeld

```
10 LET D$ = "PROGRAMMEREN"
20 FOR N = 1 TO LEN(D$)
30 PRINT MID$(D$,N,1)
40 NEXT N
50 END
RUN
P
R
O
G
R
A
M
M
E
R
E
N
```

In dit voorbeeld is de functie MID$(D$,N,1) gebruikt. Er wordt dus steeds één teken, en wel het N^{e} teken uit string D$ genomen en afgedrukt.

ASC-functie

De ASC-functie (ASC van ASCII = American Standard Code for Information

Interchange) berekent de ASCII-code van het eerste teken uit de string die als argument van de functie is opgenomen. De code wordt decimaal weergegeven. Tabel 1 toont de ASCII-coden voor de hoofdletters (voor kleine letters zijn er weer andere codes), cijfers en leestekens.
Voorbeeld:

```
NEW
10 B = ASC("XYZ")
20 A$ = "BASIC"
30 PRINT B,
40 PRINT ASC(A$)
50 END
RUN
    88   66
```

Tabel 1

ASCII (decimaal)	teken	ASCII (decimaal)	teken
65	A	10	line feed
66	B	13	return
67	C	32	spatie
68	D	33	!
69	E	34	'
70	F	35	#
71	G	36	$
72	H	37	%
73	I	38	&
74	J	39	'
75	K	40	(
76	L	41	)
77	M	42	*
78	N	43	+
79	O	44	,
80	P	45	–
81	Q	46	.
82	R	47	/
83	S	48	0
84	T	49	1
85	U	50	2
86	V	51	3
87	W	52	4
88	X	53	5
89	Y	54	6
90	Z	55	7
		56	8
		57	9

Het eerste teken van de string XYZ is X. De ASCII-code van X is 88. In regel 10 wordt deze waarde toegekend aan de variabel B en in regel 30 wordt de waarde van B afgedrukt.
In regel 40 wordt de ASCII-code van de letter B (het eerste teken van de string BASIC) bepaald. Deze code is 66.

CHR$-functie

De CHR$-functie (CHR van character = karakter) doet het omgekeerde van de

ASC-functie. Hij zet nl. de ASCII-code die als argument is gegeven om in een karakter (string van één teken).

Voorbeeld:

```
NEW
10 Z1$ = CHR$(75)
20 PRINT Z1$
30 FOR I = 65 TO 70
40 A$ = CHR$(I)
50 PRINT A$
60 NEXT I
70 END
RUN
K
A
B
C
D
E
F
```

In regel 10 wordt Z1$ gelijk aan het teken dat als ASCII-code 75 heeft. Dit teken is de letter K (zie tabel 1).
In de regels 30 t/m 60 worden de tekens afgedrukt die als ASCII-code de waarde 65 t/m 70 hebben. Dit zijn de letters A t/m F.

VAL-functie

Een string kan bestaan uit letters, leestekens en ook cijfers. Willen we met de cijfers die deel uitmaken van een string rekenen, dan moeten we ze eerst omzetten in „echte" cijfers, d.w.z. we willen werken met de *getalwaarden*. Als string zijn de cijfers in de computer nl. weergegeven volgens de ASCII-code.
Dit omzetten in echte getalwaarden kan gebeuren met de VAL-functie (VAL van value = waarde).
Stel, we willen de getalwaarde hebben van het stringteken 6 in de string BASIC6. We kunnen dit als volgt doen:

```
NEW
10 A$ = RIGHT$ ("BASIC6", 1)
20 W = VAL(A$)
30 PRINT W
40 END
RUN
      6
```

Het resultaat van de VAL-functie is 0, wanneer wordt getracht om de getalwaarde van een letter of leesteken uit een string te bepalen.

STR$-functie

De STR$-functie (STR van string) doet het omgekeerde van de VAL-functie. Hij maakt van een getal een string.

Voorbeeld:

```
NEW
10 D$ = STR$(56)
20 PRINT D$
30 END
RUN
56
```

Deze functie is vooral dan erg handig, wanneer we getallen in een bepaalde vorm willen afdrukken, bijvoorbeeld in combinatie met tekst. De computer reserveert immers voor elk getal een aantal printposities. Een getal van bijv. 2 cijfers wordt dus nooit tegen de kantlijn afgedrukt. Willen we dat nu juist wel, dan kunnen we m.b.v. de STR$-functie van het getal een string maken. Een string wordt immers wel tegen de kantlijn afgedrukt. (In bovenstaand voorbeeld de string 56.)

24. Rekenkundige functies

In hoofdstuk 18 hebben we al eens kennis gemaakt met een tweetal zgn. rekenkundige-functies, nl. INT en RND. Zo'n functie bestaat uit drie letters die aangeven om welke functie het gaat, met daarachter een getal of variabele tussen haakjes. Dit is het argument waarop de functie betrekking heeft.
In dit hoofdstuk worden de resterende functies behandeld.

SQR-functie

De SQR-functie (SQR van square root = vierkantswortel) berekent de tweedemachts wortel van het argument. Voorwaarde is, dat het argument groter dan of gelijk is aan 0.

Voorbeeld:

```
NEW
10 A = SQR(25)
20 PRINT A
30 PRINT SQR(100)
40 END
RUN
    5
    10
```

ABS-functie

De ABS-functie kunnen we gebruiken om de zgn. *absolute waarde* van een getal te bepalen, d.w.z. de waarde van een getal, ongeacht het teken. Het resultaat van de ABS-functie is dan ook altijd een positief getal.

Voorbeeld:

```
NEW
10 PRINT ABS(56)
20 PRINT ABS(-3467)
```

(vervolg op volgende pagina)

```
30 A = 37-89
40 B = ABS(A)
50 PRINT B
60 END
RUN
        56
        3467
        52
```

EXP-functie

Deze functie voert een machtsverheffing uit met als grondtal de waarde e ($\approx$ 2,71828) en als exponent het argument van de functie.
EXP(3) betekent dus: e^3.

Voorbeeld:

```
NEW
10 A = EXP(1)
20 PRINT A
30 PRINT EXP(10)
40 END
RUN
   2.71828183
   22026.4658
```

LOG-functie

De LOG-functie (LOG van logarithme) doet het omgekeerde van de EXP-functie. De LOG-functie bepaalt nl. de natuurlijke logarithme, d.w.z. de logarithme met het grondtal e, van het argument. Voorwaarde is, dat het argument groter is dan 0.
Het verband tussen de EXP-functie en de LOG-functie is als volgt:
Stel A = e^B. Dan geldt: B = ${}^e\log$ A:
Of, als BASIC expressie:
A = EXP(B) en B = LOG(A)

We kunnen ook zeggen: in B = LOG(A) berekent de LOG-functie tot welke macht het getal e moet worden verheven om A (het argument) te krijgen. Het argument mag dus niet negatief of nul zijn, omdat e tot de macht „wat-dan-ook" nooit negatief of nul kan worden (e is immers positief).

Voorbeeld:

```
NEW
10 PRINT LOG(3)
20 D = LOG(0.000001)
30 PRINT D
40 PRINT LOG(10)
50 END
RUN
 .1098612E01
-.1381551E02
 .2302585E01
```

SGN-functie

Met de SGN-functie (SGN van sign = teken) kan gemakkelijk het teken van een getal worden bepaald. Het resultaat van de SGN-functie is nl.:

- +1 wanneer het argument positief is,
- 0 wanneer het argument nul is.
- − 1 wanneer het argument negatief is.

Voorbeeld:

```
NEW
10 P = SGN(10)
15 PRINT P
20 PRINT SGN(4)
30 PRINT SGN(0)
40 PRINT SGN(-45)
50 END
RUN
1
1
0
-1
```

INT-functie

De INT-functie (INT van integer = geheel getal) zijn we al eens tegengekomen in hoofdstuk 18, waar het Master Mind-programma werd opgesteld. Deze functie bepaalt zoals gezegd het dichtstbijzijnde gehele getal dat kleiner dan of gelijk is aan het argument.

Bijvoorbeeld:

INT(5.67) = 5
INT(0.23) = 0
INT (4) = 4
INT(−5.999) = −6
INT(0) = 0
INT(1.79) = 1
enz.

Let dus goed op wanneer de INT-functie betrekking heeft op een negatief argument: INT(−5.999) = −6 en niet −5.
De functie bepaalt immers het dichtstbijzijnde gehele getal dat kleiner dan of gelijk is aan het argument; en −5 is groter dan −5.999 ......

SIN-, COS- en TAN-functie

BASIC kent drie zgn. goniometrische functies, nl. de sinus-, cosinus- en tangens-functie.
Noemen we het argument even X, dan geldt:

- SIN (X), berekent de sinus van de hoek X;
- COS (X), berekent de cosinus van de hoek X;
- TAN (X), berekent de tangens van de hoek X.

Het argument van deze goniometrische functies moet *altijd zijn uitgedrukt in radialen.* Het verband tussen radialen en hoeken is:

180° = radialen

$1° = \frac{\pi}{180°}$ radialen

andersom geldt:

1 radiaal = $\frac{180°}{\pi}$ graden

De sinus van bijv. 33 graden berekenen we dus als volgt:

10 A = SIN(33*3.1428/180)

De nauwkeurigheid kan eventueel nog worden verhoogd door voor het getal π de berekening 22/7 in te voeren:

10 A = SIN(33*22/7/180)

Wanneer deze omrekening in een goniometrische functie vaker in een programma voorkomt, doen we er goed aan om het resultaat van 22/7/180 toe te kennen aan een variabele en deze variabele in de functies te vermelden. De computer

hoeft de berekening dan slechts eenmaal uit te voeren, hetgeen uiteraard tijdbesparend werkt.

```
10 G = 22/7/180
20 A = SIN(33*G)
30 A1 = COS(45*G)
40 A2 = TAN(10*G)
```

```
10 R=22/7/180
20 FOR H=0 TO 360 STEP 12
30 PRINT SIN(H*R)
40 NEXT H
50 END
RUN
 0
 .207994147
 .40689066
 .587989832
 .743370412
 .866236075
 .951212694
 .994583404
 .994451176
 .950821793
 .8656036
 .742524027
 .586966557
 .405735253
 .206757146
-1.264489E-03
-.209230816
-.408045416
-.589012167
-.74421561
-.866867165
-.951602074
-.994714042
-.994317357
-.950429372
-.864969739
-.741676454
-.585942344
-.404579197
-.205519815
 2.52897597E-03
```

Afb. 17

In afb. 17 is weergegeven hoe de computer de sinuswaarden van de hoeken tussen 0° en 360°, in stappen van 12°, kan berekenen.

Opmerking 1:

De computer waarop dit programma is uitgevoerd laat de exponent (E..) weg wanneer deze nul is, dat wil zeggen wanneer de mantisse met 1 wordt vermenig-

vuldigd. E00 is immers 10° = 1. .207994147E00 wordt dus weergegeven als .207994147.
Bovendien worden nullen als laatste cijfers van de mantisse niet afgedrukt. Het getal .865603600 wordt weergegeven als .8656036.

Opmerking 2:
De goniometrische functies in BASIC worden berekend volgens een benaderingsformule (een reeksontwikkeling). Zo kan het dus gebeuren dat SIN(180*R), d.w.z. de sinus van 180°, wordt weergegeven als −1.264489E-03 = 0,001264489 terwijl dit eigenlijk 0 moet zijn.
(Bovendien is 22/7 een repeterende breuk, zodat SIN(180*R) niet helemaal gelijk is aan de sinus van 180°.)

DEF FN

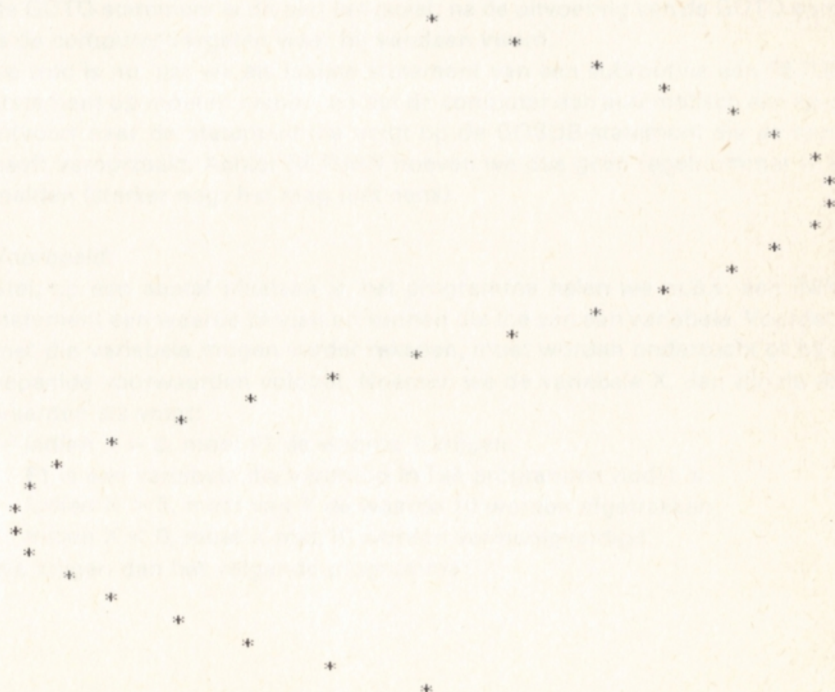
Is een bepaalde rekenkundige functie niet in BASIC aanwezig, dan hebben we nog de mogelijkheid om de functie zelf te definiëren, en wel met de opdracht DEF FN(define function = definieer functie).
Achter FN moeten we de naam van de functie aangeven. Deze naam mag slechts uit één letter bestaan, eventueel gevolgd door één cijfer. Achter die naam geven we dan het argument van de functie aan.
Als voorbeeld zullen we het resultaat van de vierkantsvergelijking $3x^2 + 2x - 4$ bepalen voor X is 0 t/m 10.

```
NEW
10 DEF FNK(X) = 3*X*X + 2*X-4
20 FOR X = 0 TO 10
30 PRINT FNK(X)
40 NEXT X
50 END
RUN    -4

         1
        12
        29
        52
        81
       116
       157
       204
       257
       316
```

TAB-functie

De TAB-functie heeft dezelfde uitwerking als de tabulator-toets op een typemachine: de afdrukkop (bij weergave op het beeldscherm is dat de cursor) gaat een op te geven aantal print-posities verder. Het argument van de TAB-functie geeft aan om hoeveel print-posities het gaat. T.g.v. de functie TAB(10) bijvoorbeeld, gaat de cursor op het scherm 10 posities naar rechts. De TAB-functie is erg handig wanneer een hoeveelheid tekst en/of getallen in een bepaalde vorm, bijv. van een tabel, moet worden weergegeven.
De functie kan ook worden gebruikt om grafieken te „construeren". Afb. 18 geeft daar een voorbeeld van; het programma geeft op het beeldscherm of op een printer de grafische voorstelling weer van de sinus-functie. De sinus-functie wordt berekend van 0° t/m 360° in stappen van 12°. Er worden dus 31 meetpunten berekend.



Afb. 18

De maximum-waarde bereikt de sinus-functie bij 90°. De functie is daar 1. De minimum-waarde is −1 en ligt bij 270°. Om de grafische voorstelling beter over het beeldscherm (of het papier van de printer) te verdelen, gaan we de sinus-waarden vermenigvuldigen met 30. De maximum- en minimum-waarde zijn dan resp. +30 en −30. Omdat de eerste print-positie op het scherm of de printer-positie 0 is, tellen we er nog eens 30 bij op. De maximum-waarde van de sinus-functie wordt dan +60; de minimum-waarde wordt 0. In BASIC geven we deze berekening weer zoals in regel 30 (afb. 18) is gedaan:

```
30 S = 30*SIN(H*G) + 30
```

S is dus het argument voor de TAB-functie in regel 40.

25. GOSUB en RETURN

Het komt vaak voor dat op verschillende plaatsen in het programma eenzelfde bewerking moet worden uitgevoerd. Dit zou dan inhouden dat we eenzelfde reeks statements meerdere malen moeten intypen en in het geheugen opslaan. Er is echter een manier om zo'n serie statements slechts eenmaal in het programma op te nemen en „aan te roepen" wanneer we ze nodig hebben. We creëren dan een zgn. *subroutine* die met een GOSUB (go to subroutine)-statement kan worden aangeroepen. Achter GOSUB vermelden we het regelnummer van de eerste statement van de subroutine.
De GOSUB-statement is, met één uitzondering, gelijk aan de GOTO-statement. Die uitzondering is dat de computer bij de uitvoering van de GOSUB-sprong *onthoudt waar hij vandaan kwam,* d.w.z. waar de GOSUB-statement stond. Bij de GOTO-statement is dit niet het geval; na de uitvoering van de GOTO-sprong is de computer vergeten waar hij vandaan kwam.
De truc is nu, dat we als laatste statement van een subroutine een RETURN-statement op moeten nemen, en dat de computer dan automatisch een sprong uitvoert naar de statement die volgt op de GOSUB-statement die de sprong heeft veroorzaakt. Achter RETURN hoeven we dus geen regelnummer te vermelden (sterker nog: het mag niet eens).

Voorbeeld:
Stel, op een aantal plaatsen in het programma halen we m.b.v. een INPUT-statement een waarde binnen en kennen die toe aan een variabele. Voordat we met die variabele mogen verder rekenen, moet worden onderzocht of hij aan bepaalde voorwaarden voldoet. Noemen we de variabele X, dan zijn de voorwaarden als volgt:

- Indien X = 0, moet F1 de waarde 1 krijgen.
  F1 is een variabele die verderop in het programma nodig is;
- Indien X > 5, moet van X de waarde 10 worden afgetrokken;
- Indien X < 0, moet X met 10 worden vermenigvuldigd.

We krijgen dan het volgende programma:

```
10 INPUT A
20 LET X = A
30 GOSUB 500
40 LET A = X
.
.
.
120 INPUT B
130 LET X = B
140 GOSUB 500
150 LET B = X
.
.
.
500 F1 = 0
510 IF X = 0 THEN F1 = 1
520 IF X>5 THEN X = X-10
530 IF X<0 THEN X = X*10
540 RETURN
```

In regel 10 krijgt A een waarde die via het toetsenbord wordt ingevoerd. In regel 20 kennen we die waarde toe aan de variabele X, omdat de subroutine, gevormd door de regels 500 t/m 540 met die variabele werkt.
De statement op regel 30 roept de subroutine aan, d.w.z. er volgt een sprong naar regel 500 en de computer onthoudt het regelnummer van deze GOSUB-statement (regel 30).
Nu wordt de subroutine uitgevoerd. Eerst wordt F1 nul gemaakt. Daarna wordt onderzocht of X gelijk is aan nul. Is dit het geval, dan krijgt F1 de waarde 1. Is dit niet het geval, dan blijft F1 nul en wordt verder gegaan bij regel 520. Van X wordt 10 afgetrokken als X groter is dan 5; X wordt met 10 vermenigvuldigd als X kleiner is dan nul.
Tenslotte komt de computer bij regel 540. Er volgt nu een sprong naar de statement die volgt op die statement waarvan de computer het regelnummer in zijn geheugen heeft opgeslagen, d.w.z. naar de statement die volgt op regel 30. In regel 40 krijgt A dus de nieuwe waarde.

Precies hetzelfde gebeurt in de regels 120 t/m 150, nu echter met de variabele B.
In regel 130 krijgt X dezelfde waarde als B. Daarna wordt de subroutine opnieuw aangeroepen, waarbij de computer regelnummer 140 onthoudt. Op deze regel staat immers de GOSUB-statement. Na uitvoering van de subroutine zorgt de RETURN-statement ervoor dat wordt teruggekeerd naar de statement die volgt op regel 140, d.w.z. naar regel 150.

Subroutines mogen worden „genest", d.w.z. vanuit een subroutine mag naar een andere subroutine worden gesprongen, vanuit die subroutine mag weer naar een andere subroutine worden gesprongen, enz. De enige beperkende factor is de capaciteit van het geheugen waarin de „terugkeer-regelnummers" worden opgeslagen.

Plaats alle subroutines bij voorkeur aan het einde van het hoofdprogramma, zodat u er een goed overzicht over heeft. Zorg er ook voor dat de laatste statement van het hoofdprogramma een END-statement is, zodat de computer, na uitvoering van het hoofdprogramma, niet nog eens met de subroutines begint. In dat geval ziet de computer nl. op een gegeven moment een RETURN-statement die niet is voorafgegaan door een GOSUB-statement. Er volgt dan een foutmelding.

26. PEEK en POKE

We hebben gezien dat we met de LET- en INPUT-statements een waarde kunnen toekennen aan een variabele. Die variabele is eigenlijk het adres van een geheugenlokatie in de computer. Welke geheugenlokatie dat is weten we niet; de computer zoekt zelf een vrije lokatie op wanneer hij de variabele voor het eerst in het programma tegenkomt.
Willen we een geheugenlokatie aanspreken met een door ons op te geven adres, dan moeten we gebruik maken van de PEEK- en POKE-statements. Met POKE kunnen we een geheugenlokatie vullen; met PEEK kunnen we een geheugenlokatie uitlezen.

Vooral de POKE-statement is erg handig, omdat bij de meeste computers het beeldschermgeheugen deel uitmaakt van het werkgeheugen van de computer. In het beeldschermgeheugen staan de codes van de karakters die op het beeldscherm worden weergegeven. Het aantal geheugenlokaties van het beeldschermgeheugen is evengroot als het aantal karakters dat op het beeldscherm kan worden weergegeven. Heeft het beeldscherm bijv. een indeling van 24 regels × 40 karakters, dan is het beeldschermgeheugen 24 × 40 = 960 lokaties groot.
Met de POKE-statement is het mogelijk om een lokatie van het beeldschermgeheugen te vullen met de code voor een bepaald karakter, meestal zijn dat ASCII-codes. Elke lokatie in het beeldschermgeheugen komt overeen met een bepaalde plaats op het beeldscherm. De eerste lokatie komt overeen met de linkerbovenhoek, de tweede lokatie met het tweede karakter op de eerste regel, enz. Bij een 24 × 40 beeldscherm komt de 25e lokatie overeen met het eerste karakter op de tweede regel. De laatste geheugenlokatie van het beeldschermgeheugen komt overeen met het laatste karakter op de laatste regel.

Voorbeeld:
Stel het beeldschermgeheugen van een computer loopt van adres 25000 t/m 25959. We willen op de eerste, tweede en derde regel op de 4e karakterpositie de letter A weergeven (de ASCII-code van A is 65):

```
10 POKE 25003,65
20 POKE 25043,65
30 POKE 25083,65
40 END
```

Achter POKE geven we eerst het adres aan van de te vullen geheugenlokatie. Daarna vermelden we de waarde waarmee die lokatie moet worden gevuld. Beide waarden moeten in het decimale talstelsel worden opgegeven.

Met de PEEK-statement kunnen we de inhoud van een geheugenlokatie uitlezen en toekennen aan een variabele:

```
10 A  = PEEK (10000)
20 C  = PEEK (25049)
30 Z5 = PEEK (45639)
```

enz.

In regel 10 wordt A gelijk aan de inhoud van geheugenlokatie 10000. Daarna krijgt C een waarde die gelijk is aan de inhoud van lokatie 25049 (in ons vorige voorbeeld zou C dus gelijk worden aan de code van het karakter dat op de 10e positie van de tweede regel op het beeldscherm wordt weergegeven). In regel 30 tenslotte wordt Z5 gelijk aan de inhoud van lokatie 45639.

PEEK- en POKE statements worden ook vaak gebruikt om resultaten en parameters uit te wisselen tussen het BASIC-programma en een in machinetaal geschreven programma.

27. ON...GOTO

Achter deze statement kan een aantal regelnummers worden opgegeven. Afhankelijk van de waarde van een variabele wordt dan naar één van die regelnummers gesprongen.
Zo heeft de statement:

20 ON K GOTO 100, 110, 120, 130

een sprong tot gevolg naar het K^e opgegeven regelnummer, d.w.z. als K = 1 volgt een sprong naar regel 100, als K = 2 naar regel 110, als K = 3 naar regel 120 en als K = 4 naar regel 130.
Uiteraard moet K aan bepaalde voorwaarde voldoen:
1. K mag niet negatief zijn;
2. K mag max. 255 zijn.

Is aan een van deze voorwaarden niet voldaan, dan geeft de computer een foutmelding. Is K = 0 of is K groter dan het aantal opgegeven regelnummers, dan wordt verdergegaan bij de statement die volgt op de ON...GOTO-statement.
De statement:
10 ON K GOTO 100, 110, 120, 130

komt dus overeen met:
10 IF K = 1 THEN 100
20 IF K = 2 THEN 110
30 IF K = 3 THEN 120
40 IF K = 4 THEN 130
Is K geen geheel getal, dan haalt de computer eerst het gedeelte achter de komma weg, d.w.z. hij voert eerst de berekening INT(K) uit.

28. REM

M.b.v. de REM-statement hebben we de mogelijkheid om commentaar aan het programma toe te voegen. De computer zelf doet met de REM-statement niets, hoewel hij bij het opvragen van een listing wel wordt weergegeven.

Voorbeeld:

```
10 REM DIT PROGRAMMA GENEREERT
20 REM GETALLEN TUSSEN 0 en 20
30 A = INT(RND(1)*19) + 1
40 REM WEERGAVE OP SCHERM
50 PRINT A
60 REM VOLGENDE GETAL
70 GOTO 20
RUN
        6
       17
        5
        2
```

enz.

Zoals u ziet wordt de tekst achter REM niet tussen aanhalingstekens vermeld en is het mogelijk om naar een REM-statement te springen, zoals in regel 70 gebeurt.

29. Korter maken van een programma

We hebben nu de BASIC-statements behandeld zoals die op de meeste computersystemen kunnen worden uitgevoerd.
Als laatste zullen we enkele programmeertrucs behandelen om een programma korter te maken, zowel in uitvoeringstijd als in geheugenruimte die in beslag wordt genomen.

Meer statements op een regel

Het is mogelijk om op één regel meerdere statements op te nemen. Die statements moeten dan van elkaar worden gescheiden m.b.v. een dubbele punt (bij sommige computers m.b.v. een schuine streep /, of met een punt-komma).

Voorbeelden:

```
50 FOR N = 1 TO 10 : READ A(N) : NEXT N
260 IF H = 9 THEN PRINT „FOUT" : K = K + 1 : GOTO 130
270 INPUT A$ : IF A$ <> „JA" THEN 270
```

Bij de IF-Statement moet u even opletten; het gedeelte achter THEN wordt alleen uitgevoerd als aan de gestelde voorwaarde is voldaan. In regel 260 uit bovenstaand voorbeeld worden dus, wanneer H *on*gelijk is aan g, *geen* van de drie statements achter THEN uitgevoerd, maar wordt direct verdergegaan bij de statement die volgt op regel 260.

Weglaten van spaties

In alle voorgaande programma's hebben we tussen de afzonderlijke delen van een statement een spatie ingetypt. Dit hebben we echter alleen gedaan om de leesbaarheid van de programma's te vergroten. De BASIC-interpreter heeft deze spaties voor het herkennen en juist uitvoeren van een statement helemaal niet nodig. Elke spatie wordt wel opgeslagen in het geheugen van de computer en neemt daar één geheugenlokatie in beslag.

Verwijder dus alle spaties uit het programma indien u geheugenruimte wilt besparen. Hetzelfde geldt uiteraard voor REM statements.

Voorbeeld:
de statement:

100 PRINT A, B, C

neemt drie geheugenlokaties meer in beslag dan:

100 PRINTA,B,C

De spaties die tussen het regelnummer en de eerste letter van een statement worden ingetypt, worden door de computer genegeerd en dus ook niet in het geheugen opgeslagen.

Gebruik variabelen in plaats van constanten

In hoofdstuk 12 hebben we gezien dat de computer decimale breuken volgens een speciale methode weergeeft en verwerkt. Voeren we bijv. in 456.34, dan maakt de computer daar .4563400 E 03 van.
Wanneer het getal 456.34 nu meerdere malen in een programma voorkomt, dan moet de computer dat getal steeds eerst omrekenen in een voor hem geschikte vorm. Dit neemt onnodig veel tijd en geheugenruimte in beslag. In zo'n geval moeten we aan het begin van het programma de constante toekennen aan een variabele en in het vervolg van het programma met die variabele verderrekenen.
Komt de constante 56.099 bijvoorbeeld 10 maal in een programma voor, dan schrijven we aan het begin van het programma:

10 LET C = 56.099

en werken verder met de variabele C.

Meervoudig gebruik van een variabele

Wanneer aan het begin van het programma gebruik is gemaakt van bijv. de variabele N, dan mag deze variabele verderop in het programma best nog eens worden gebruikt. U bespaart op die manier geheugenruimte die anders voor een tweede variabele nodig zou zijn.

Voorbeeld:

```
10 FOR N=1 TO 25
20 READ F(N):NEXT N
30
 .
 .
 .
```

(vervolg op volgende pagina)

```
345 LET N=P+4
350 IF N=9 THEN 670
.
.
```

De regels 10 en 20 vormen een programmalus waarin N varieert. Wanneer de lus is doorlopen en het programma op een gegeven moment verdergaat bij regel 30, is N in feite niet meer nodig. De variabele heeft zijn werk gedaan en de letter N is weer vrij om voor een andere variabele te worden gebruikt. Dat gebeurt in regel 345.

Vragen:

1. Hoeveel geheugenlokaties worden met DIM Z(10,10) gereserveerd:
2. Wat wordt t.g.v. onderstaand programma op het beeldscherm (of de printer) weergegeven?

   ```
   10 P$ = "1"
   20 Q$ = "2"
   30 R$ = P$ + Q$
   40 PRINT R$
   50 END
   RUN
   ```
3. Wat wordt t.g.v. onderstaand programma weergegeven?

   ```
   10 LET F1$ = "PRINT"
   20 FOR I - 1 TO LEN (F1$)
   30 PRINT MID$ (F1$, I,1);
   40 NEXT I
   50 END
   60 RUN
   ```
4. Wat is het resultaat van :

   ABS (INT(−4.5))

   en van:

   INT(ABS(−4.5))

   5. Voor welke waarden van T is

   INT(T/2) = T/2?

Antwoorden:

1. Er wordt geheugenruimte gereserveerd voor een tabel, bestaande uit 11 rijen en 11 kolommen (inclusief rij 0 en kolom 0). In totaal: 11 × 11 = 121 geheugenlokaties.
2. R$ is de „samenvoeging" van de string „1" en de string „2".
   R$ is dus 12.
3. LEN (F1$) is gelijk aan 5. Regel 20 wordt dus: FOR I = 1 TO 5. De 5 letters van het woord PRINT wordt in regel 30 achtereenvolgens afgedrukt, en dat gebeurt op dezelfde regel

t.g.v. de punt-komma aan het einde van regel 30. Resultaat is dan ook dat het woord PRINT verschijnt.

4. ABS(INT(−4.5)) = ABS(−5) = 5
   INT(ABS(−4.5)) = INT(4.5) = 4
5. INT(T/2) = T/2 als T even is.
   Dan is nl. T/2 een geheel getal.

Bij Kluwer Technische Boeken zijn de volgende computerboeken verschenen:

ISBN			
90 201 1127 2	Aspinall/Daglass	Inleiding tot microprocessors	ing.
90 201 1305 4	Boon	Basic en huiscomputers	geb.
90 201 1398 4	Dirksen	Microcomputers	geb.
90 201 1332 1	Dirksen	Computeroriëntatie	ing.
90 201 0711 9	Heyer den/ Engelsen den	Computers aan het werk	ing.
90 201 1095 0	Klein	Microcomputer-systemen	ing.
90 201 1331 3	Lörincz	Fortran V	ing.
90 201 1254 6	Pelka	Wat is een microprocessor?	ing.
90 201 1257 0	Prooijen v.	Basic voor beginners	ing.
90 201 1385 2	Wilmink/Boon	Microcomputergids (jaarlijkse uitgave)	ing.
90 201 0992 8	Winia	Wegwijs in wetenschappelijke calculators	ing.
90 201 1010 1	Winia	Wegwijs in programmeerbare rekenapparaten	ing.
90 201 1128 0	Zaks	(Micro)computers voor hobby en werk	ing.
90 201 1129 9	Zaks	Microprocessors; van chip tot systeem	ing.
90 201 1054 3	Zuiderveen	Digitale schakelingen, deel 1 t/m 3	geb.

De programmeertaal BASIC is tegen het eind van de jaren 60 ontwikkeld aan het Dartmouth College in de Verenigde Staten, en was in eerste instantie bedoeld voor onderwijsdoeleinden. De taal bleek echter voor zoveel toepassingen geschikt dat ze inmiddels is uitgegroeid tot een van de meest gebruikte programmeertalen.
Omdat BASIC vrij eenvoudig van opbouw is, beslaat het vertaalprogramma dat de instructies vertaalt in computercodes relatief weinig geheugenruimte, en juist hierdoor kan BASIC zich verheugen in een enorme populariteit onder de microcomputergebruikers.
Dit boek is bestemd voor degenen die direct aan de slag willen; geen ellenlange inleidingen over de werking van de computer of over moeilijke programmeertechnieken, maar direct de behandeling van de BASIC-statements, afgewisseld met programmavoorbeelden.

ISBN 90 201 1257 0